$2.95

COMPUTER LANGUAGE™

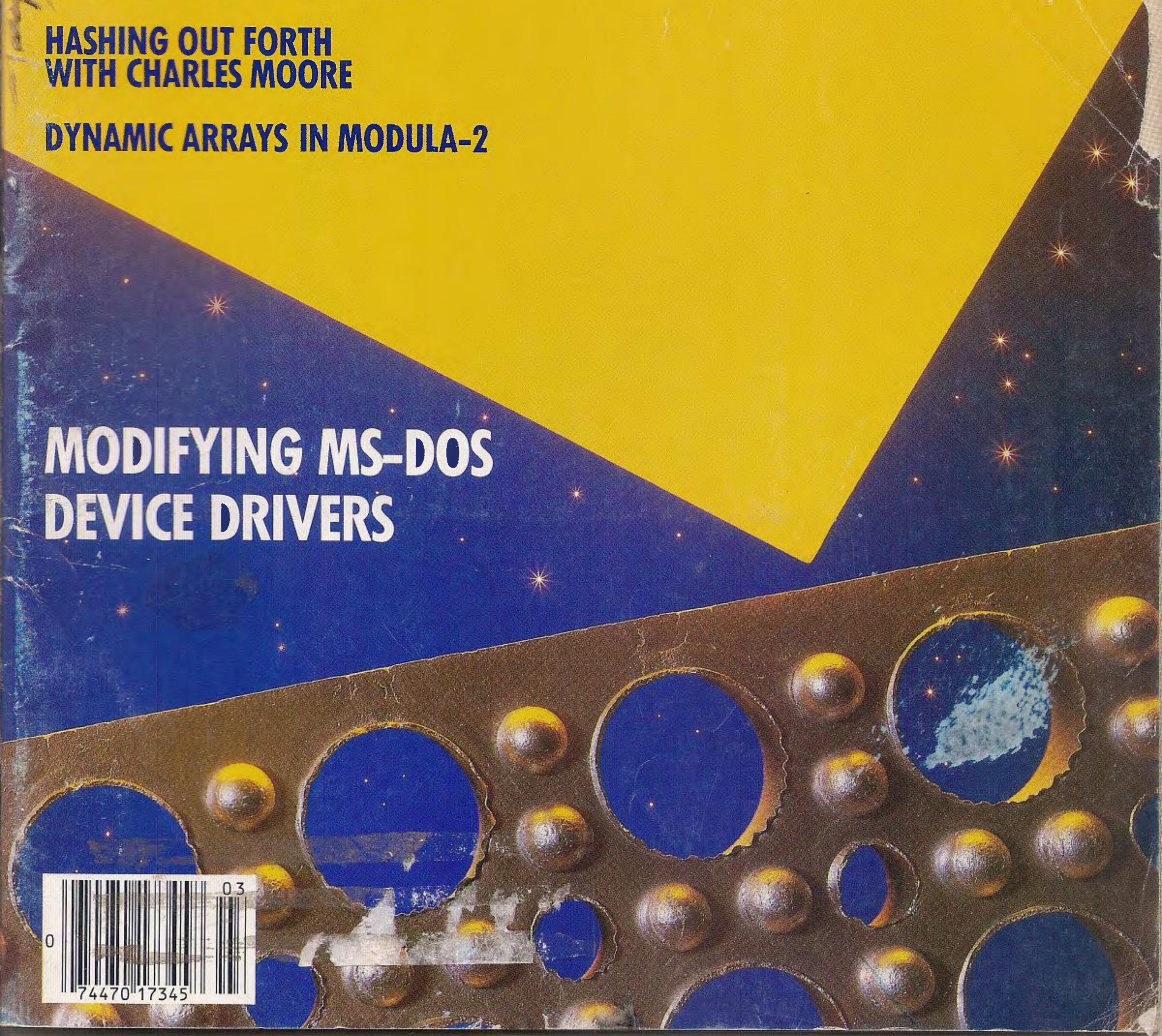
VOLUME 2, NUMBER 3

MARCH 1985

SOL LIBES COVERS MULTIPROCESSING SYSTEMS

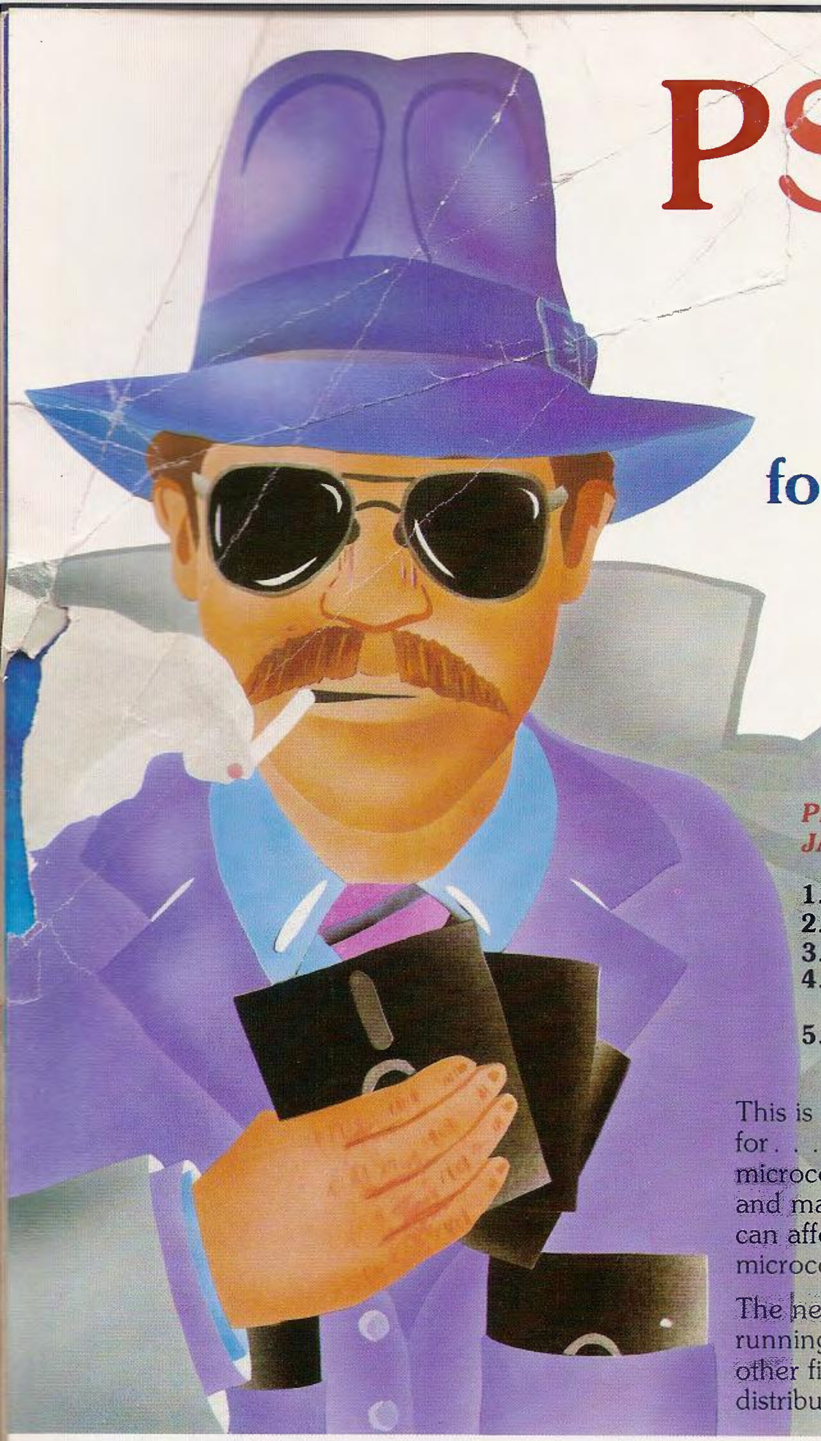
HASHING OUT FORTH WITH CHARLES MOORE

DYNAMIC ARRAYS IN MODULA-2



MODIFYING MS-DOS DEVICE DRIVERS

03
0
74470 17345



PSSST...

JANUS/Ada

for $99.95!!

PRESENTING THE NEW JANUS/Ada C-PAK!!

1. **Janus/Ada Compiler**
2. **Janus/Ada Linker**
3. **Janus/Ada Libraries**
4. **Janus/Ada Example/ Programs**
5. **Janus/Ada User Manual**

AND THESE ADDED FEATURES!!

1. **Free User's Group**
2. ***$99.95 Discount* on the Janus/Ada D-Pak**
3. **No License!!**
4. **No Copy Protection!!!**
5. **Customer tested for over 3 years!!!**

This is the introductory Ada™ package you've been waiting for . . . over three years of actual field use, specifically on microcomputers, by the government, Fortune 500 businesses and major universities. Realistically priced, at $99.95, so *you* can afford the most popular Ada implementation used on microcomputers!

The new "C"-Pak is available for most microcomputers running MS-DOS, including the IBM PC AT™, as are all the other fine Janus/Ada programs. Call us or an authorized distributor for your copy today!

National Distributors

Westico, Inc.
25 Van Zant St.
Norwalk, CT 06855
(203) 853-6880

ASH II
7407 Marisol
Houston, TX 77083
(713) 933-1828

A.O.K. Computers
816 Easley St., Suite 615
Silver Springs, MD 20910
(301) 588-8446

Trinity Solutions
5340 Thornwood Dr., Suite 102
San Jose, CA 95123
(408) 226-0170

MicroProgramming, Inc.
P.O. Box 3356
Chatsworth, CA 91313
(818) 993-6475

International Distributors

Ada Australia
218 Lutwyche Rd.
Windsor 4030
QLD. Australia
(07) 57 9997

Progesco
155, rue du Faubourg
St. -Denis
75010 Paris
France
(1) 205.39. 47

Lifeboat, Inc. Japan
3-6, Kando-Nishikicho
Chiyoda-ku
Tokyo 101, JAPAN
03-293-4711

CP/M, CP/M-86, CCP/M-86 are trademarks of Digital Research, Inc.
*ADA is a trademark of the U.S. Department of Defense
MS-DOS is a trademark of Microsoft

©Copyright 1984 RR Software

RR SOFTWARE, INC.

specialists in state of the art programming

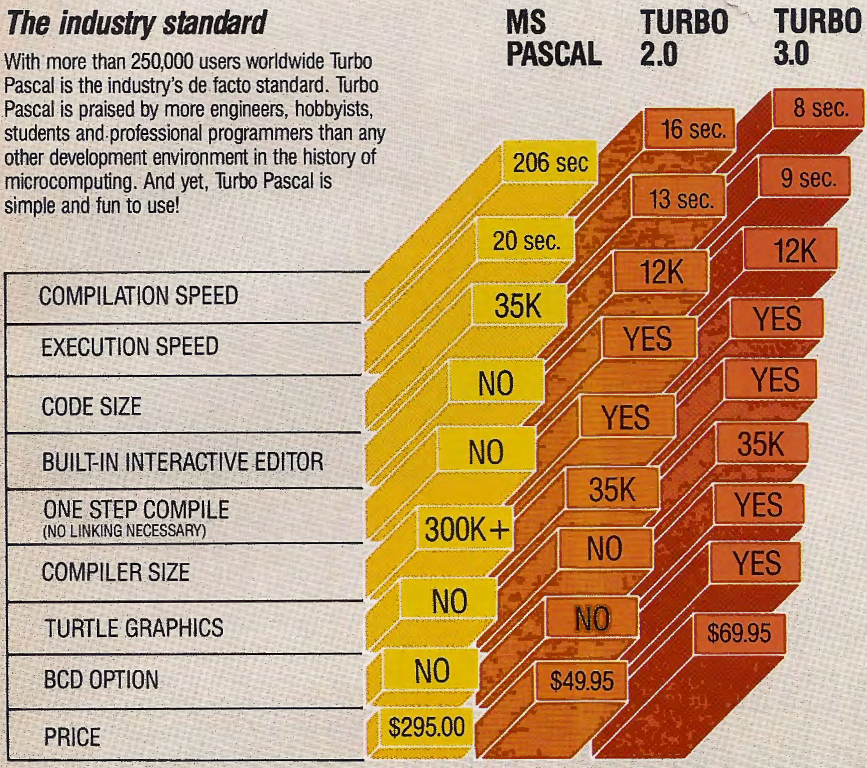
P.O. Box 1512 Madison, Wisconsin 53701
(608) 244-6436 TELEX 4998168

CIRCLE 58 ON READER SERVICE CARD

They said it couldn't be done. Borland Did It. Turbo Pascal 3.0

The industry standard

With more than 250,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use!



	MS PASCAL	TURBO 2.0	TURBO 3.0
COMPILATION SPEED	206 sec	16 sec.	8 sec.
EXECUTION SPEED	20 sec.	13 sec.	9 sec.
CODE SIZE	35K	12K	12K
BUILT-IN INTERACTIVE EDITOR	NO	YES	YES
ONE STEP COMPILE (NO LINKING NECESSARY)	NO	YES	YES
COMPILER SIZE	300K+	35K	35K
TURTLE GRAPHICS	NO	NO	YES
BCD OPTION	NO	NO	YES
PRICE	$295.00	$49.95	$69.95

The best just got better: Introducing Turbo Pascal 3.0

We just added a whole range of exciting new features to Turbo Pascal:

- First, the world's fastest Pascal compiler just got faster. Turbo Pascal 3.0 compiles twice as fast as Turbo Pascal 2.0! No kidding.
- Then, we totally rewrote the file I/O system, and we also now support I/O redirection.
- For the IBM PC versions, we've even added "turtle graphics" and full tree directory support.
- For all 16 Bit versions, we now offer two additional options: 8087 math coprocessor support for intensive calculations and Binary Coded Decimals (BCD) for business applications.
- And much much more.

The Critics' Choice

Jeff Duntemann, PC Magazine: *"Language deal of the century . . . Turbo Pascal: It introduces a new programming environment and runs like magic."*

Dave Garland, Popular Computing: *"Most Pascal compilers barely fit on a disk, but Turbo Pascal packs an editor, compiler, linker, and run-time library into just 29K bytes of random-access memory."*

Jerry Pournelle, BYTE: *"What I think the computer industry is headed for: well documented, standard, plenty of good features, and a reasonable price."*

Portability

Turbo Pascal is available today for most computers running PC DOS, MS DOS, CP/M 80 or CP/M 86. A XENIX version of Turbo Pascal will soon be announced, and before the end of the year, Turbo Pascal will be running on most 68000 based microcomputers.

An Offer you Can't Refuse

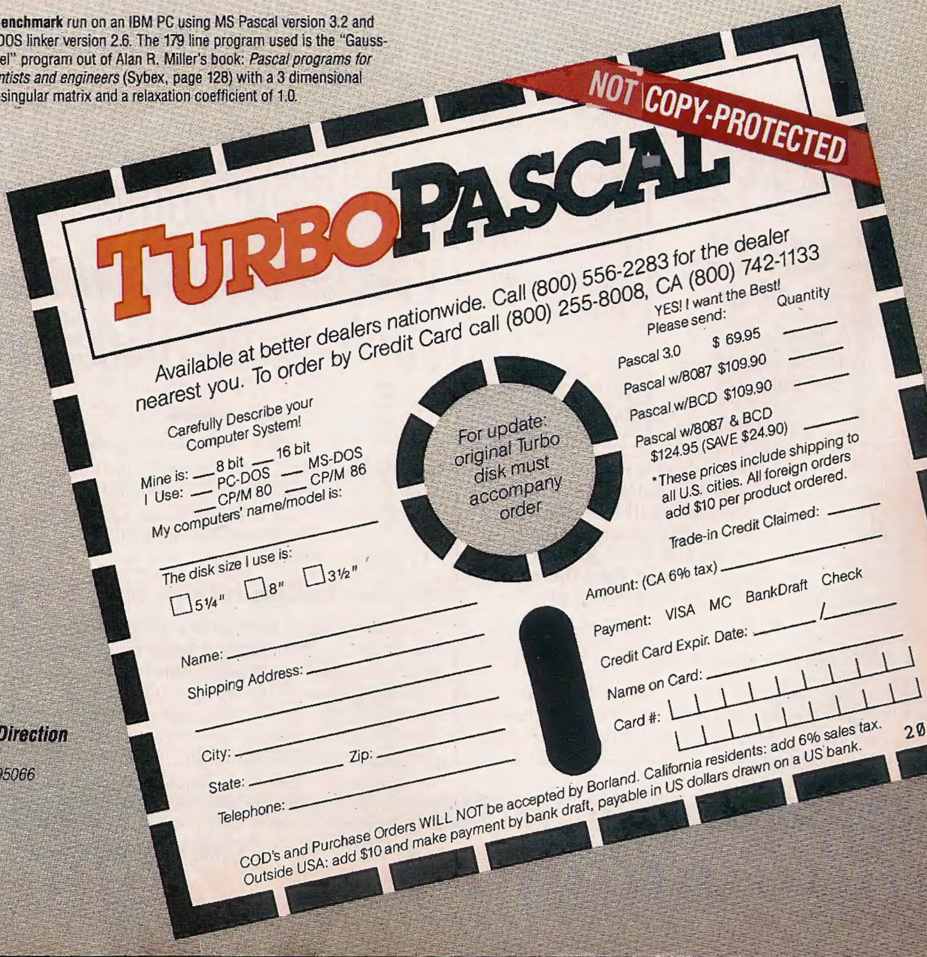
Until June 1st, 1985, you can get Turbo Pascal 3.0 for only $69.95. Turbo Pascal 3.0, equipped with either the BCD or 8087 options, is available for an additional $39.95 or Turbo Pascal 3.0 with both options for only $124.95. As a matter of fact, if you own a 16 Bit computer and are serious about programming, you might as well get both options right away and save almost $25.

Update policy

As always, our first commitment is to our customers. You built Borland and we will always honor your support.

So, to make your upgrade to the exciting new version of Turbo Pascal 3.0 easy, we will accept your original Turbo Pascal *disk* (in a bend-proof container) for a trade-in credit of $39.95 and your Turbo87 original *disk* for $59.95. This trade-in credit may only be applied toward the purchase of Turbo Pascal 3.0 and its additional BCD and 8087 options (trade-in offer is only valid directly through Borland and until June 1st, 1985).

(*) **Benchmark** run on an IBM PC using MS Pascal version 3.2 and the DOS linker version 2.6. The 179 line program used is the "Gauss-Seidel" program out of Alan R. Miller's book: *Pascal programs for scientists and engineers* (Sybex, page 128) with a 3 dimensional non-singular matrix and a relaxation coefficient of 1.0.



NOT COPY-PROTECTED

TURBO PASCAL

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

Carefully Describe your Computer System!

Mine is: ___ 8 bit ___ 16 bit
I Use: ___ PC-DOS ___ MS-DOS
___ CP/M 80 ___ CP/M 86
My computers' name/model is: ___

The disk size I use is:
☐ 5¼" ☐ 8" ☐ 3½"

For update: original Turbo disk must accompany order

YES! I want the Best!
Please send:

		Quantity
Pascal 3.0	$ 69.95	___
Pascal w/8087	$109.90	___
Pascal w/BCD	$109.90	___
Pascal w/8087 & BCD	$124.95 (SAVE $24.90)	___

*These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

Trade-in Credit Claimed: ___
Amount: (CA 6% tax) ___
Payment: VISA MC BankDraft Check
Credit Card Expir. Date: ___/___
Name on Card: ___
Card #: ___

Name: ___
Shipping Address: ___
City: ___
State: ___ Zip: ___
Telephone: ___

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

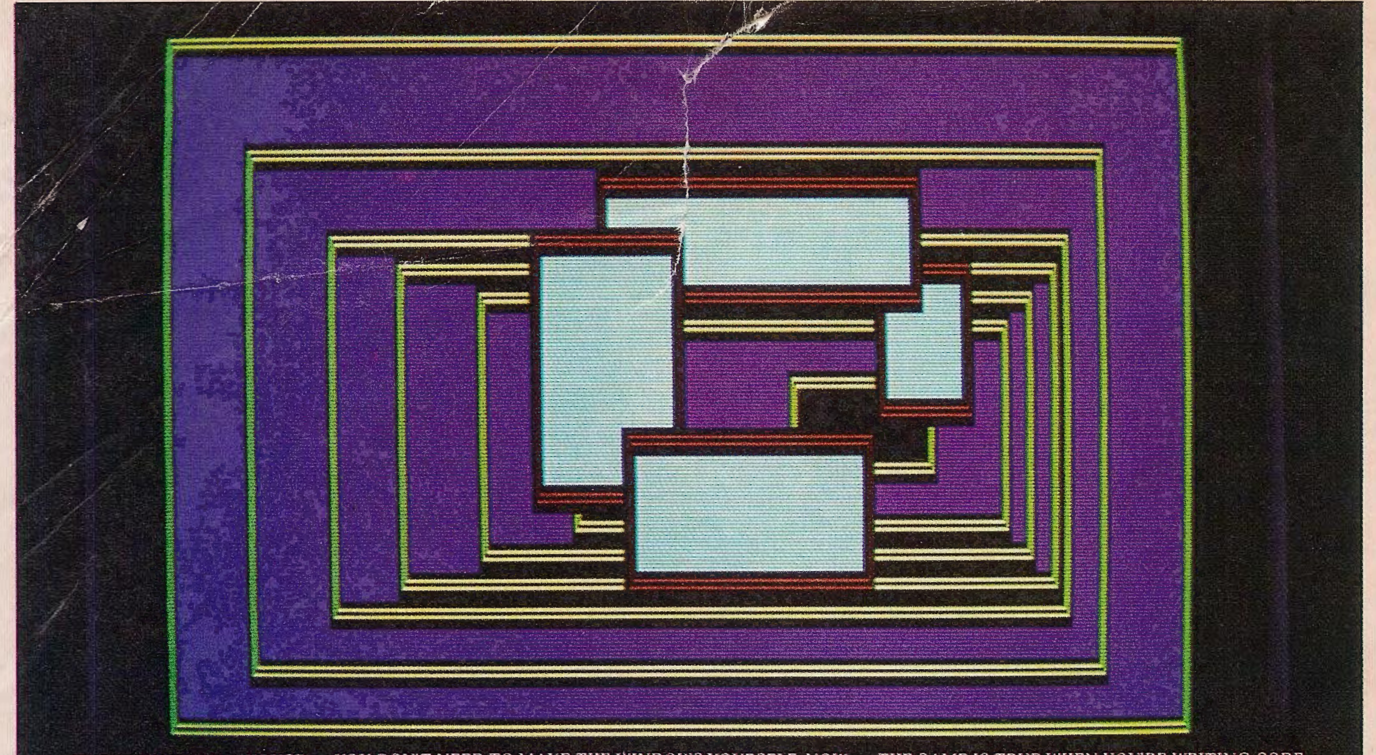
20

BORLAND INTERNATIONAL

Software's Newest Direction
4113 Scotts Valley Drive
Scotts Valley, California 95066
TELEX: 172373

Turbo Pascal is a registered trademark of Borland International, Inc.

CIRCLE 10 ON READER SERVICE CARD



WHEN YOU BUILD A HOUSE . . . YOU DON'T NEED TO MAKE THE WINDOWS YOURSELF. NOW . . . THE SAME IS TRUE WHEN YOU'RE WRITING CODE.

Why did Simon & Schuster, 3Com, Tymshare, and Revlon choose VSI – The Window Machine?

(and how come you can buy it for such a low price?)

$59.95

Windows With A View Toward The Future

The Window Machine™ occupies only 12K! Written in tight, fast Assembler, it performs like a racing engine . . . with more power than you'll probably ever need. Yet, it's an engine designed to fit in the vehicle of your choice . . . from a "stripped-down" 128K IBM PC to a fully loaded AT. The programs you write today will run on the broadest range of machines possible . . . now, and in the future.

These are coders' windows . . . designed to be built into the programs you are writing. They can overlap, move anywhere on the screen, grow, shrink, vanish or blink. They can be bordered in anything from a simple line to flashing asterisks . . . or even no border at all. And you can have up to 255 of them at a time! Color or monochrome . . . of course!

Windows Bigger Than Your Screen?

Here's where the VSI part of our name fits in. VSI means Virtual Screen Interface. Behind each window, there's a much bigger picture. VSI defines virtual screens rather than just windows. The window itself shows whatever portion of its virtual screen you wish to exhibit at any given point in your program. Each screen can be up to 128 x 255 (columns x rows, or rows x columns). And there are more than 100 screen primitives at your command.

Multilingual Windows

You can order The Window Machine with the language interface of your choice: C, Pascal, Compiled Basic, Fortran, Cobol, or PL1. We've even recently completed an interface for Turbo Pascal*, so that now true, full-featured windowing can be utilized with this fine compiler. (Turbo's own built-in "windowing" procedure is extremely limited).

Windows That Won't Break You

We decided to save you a lot of money. So, we left behind fancy binders, monogrammed slip cases and plastic presentation boxes. Instead, you'll find an extremely powerful tool and a 200 page manual written with an eye toward simplicity, clarity and completeness. (We figured if you wanted ribbons and bows you could always add them yourself.)

And by offering you the product ourselves, we were able to cut out all the middlemen and save you a tremendous amount of money.

---

*Turbo Pascal is a Trademark of Borland International

VSI *The Window Machine™*

Available for the IBM PC, XT, AT, IBM Compatibles, and the Wang, T.I., HP 150.

The Window Machine Includes:

- *Zoom Windows*
- *Multiple Virtual Screens (up to 255)*
- *Choice of Borders (including flashing borders)*
- *Support for all Color and Monochrome Video Attributes (no graphics card required)*
- *Built-in Diagnostics*
- *And much, much more*

ORDER YOUR COPY OF VSI—THE WINDOW MACHINE TODAY
For Visa and MasterCard orders call toll free:

1-800 742-2500

The Window Machine™ $59.95 Shipping and handling included
LANGUAGE INTERFACE:
☐ Lattice C ☐ Realia Cobol ☐ Microsoft Basic Compiler ☐ Microsoft Fortran ☐ PL1 ☐ Microsoft Pascal ☐ Turbo Pascal (full featured true windowing)
COMPUTER ________
Name ________
Address ________
City ________ State ____ Zip Code ________
☐ Check ☐ Money Order ☐ VISA ☐ MasterCard
Card # ________ Exp. Date ________
*California residents: tax included. Orders outside the USA: please add $5 for shipping and handling.
CL

AMBER SYSTEMS
1171 S. Saratoga-Sunnyvale Road
San Jose, CA 95129

AMBER SYSTEMS, INC. 1171 S. Saratoga-Sunnyvale Road, San Jose CA 95129

FOR DEALER INQUIRIES: CALL OUR 800 NUMBER

CIRCLE 2 ON READER SERVICE CARD

COMPUTER LANGUAGE

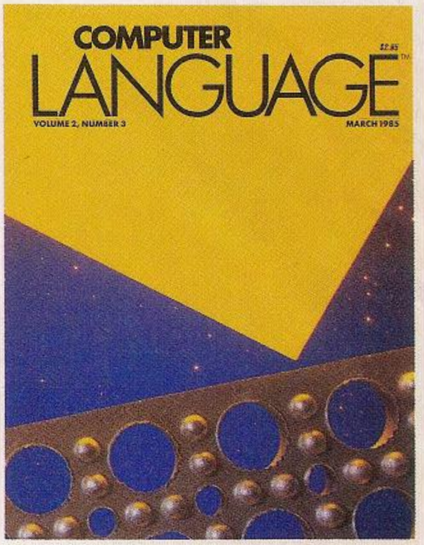
ARTICLES

Modifying MS-DOS Device Drivers **27**
by Mike Higgins
Anyone mildly proficient at 8088 assembly programming can learn to write his or her own device driver to add peripheral devices or to improve an existing driver. Since MS-DOS is one of the first widely distributed microcomputer operating systems to offer the power of customized, loadable drivers, you can learn to isolate device-specific portions of this code to make your programs more transportable to a wider variety of MS-DOS systems and also have greater control over your programming environment.

Programming Macros in C, Part II **37**
by Alexander B. Abacus
Good programmers, this author claims, plan for the debugging process during program development and treat debugging statements as a permanent part of the program by writing them before the first compilation is attempted. In this second installment of a three-part series on using C macros, we explore how macros can be used effectively in the planning, writing, and maintaining stages of the debugging process.

Trends in Multiprocessing Systems **45**
by Sol Libes
This month we have a special visit from Sol Libes, founder and ex-editor of *Microsystems* magazine. In a way not unlike his previous News and Views industry analysis column, Libes explores the trend toward multiuser and multiprocessing systems using 8- and 16-bit devices. These systems employ multiple processors to form a resource pool orchestrated by either a common operating system or other possible configurations.

Dynamic Multidimensional Arrays in Modula-2 **52**
by Namir Clement Shammas
Since Modula-2 allows the development of general purpose library modules that can be compiled separately, you can learn to extend the language in various ways. In this article we take a look at how you can develop code that will implement a feature that is missing from most Modula-2 compilers: dynamic multidimensional arrays.



DEPARTMENTS

Editor's Notes ________ **5**
Feedback ________ **7**
Industry Insight ________ **11**
CrossXthoughts ________ **13**
Mathematical parsers and interpreters
Designers Debate ________ **19**
Hashing out Forth with Charles Moore
Public Domain Software Review ________ **63**
Disk and library utilities
The Code Swap Shop ________ **67**
Product Bingo ________ **69**
Software Review ________ **70**
ProCode International's Waltz LISP
Advertiser Index ________ **80**

SQUEEZE MORE OUT OF EVERY ON-LINE MINUTE.

CompuServe Software
VIDTEX
CompuServe

WITH NEW VIDTEX™ COMMUNICATIONS SOFTWARE FROM COMPUSERVE.

Presenting the software package that makes your computer more productive and cost-efficient.

CompuServe's new Vidtex™ is compatible with many personal computers sold today (including Apple,® Commodore,® and Tandy/Radio Shack® brands). And it offers the following features*–and more–to let you communicate more economically with most time-sharing services (including CompuServe's Information Service).

Auto-Logon. Lets you log on to a host simply and quickly by utilizing prompts and responses defined by you. Also allows quick transmission of predefined responses to host application programs after logging on.

Function Keys. Let you consolidate long commands into single keystrokes. Definitions can be saved to and loaded from disk file, allowing multiple definitions for multiple applications.

Error-Free Uploading and Downloading. CompuServe "B" Protocol contained in Vidtex lets you transfer from your computer to CompuServe and from CompuServe to your computer anywhere in the country. Also provides error-free downloading from CompuServe's extensive software libraries.

Full Printer Support. Printer buffer automatically buffers characters until printer can process; automatically stops on-line transmission when full; and automatically resumes transmission when capacity is re-established. Also, lets you print contents of textual video screen or RAM buffer at any time.

Capture Buffer. Saves selected parts of a session. Contents can be written to a disk file; displayed both on and off line; loaded from disk; and transmitted to the host.

On-line Graphics. Integral graphics protocol displays stock charts, weather maps and more.

If you are already a CompuServe subscriber, you can order Vidtex on line by using the GO ORDER command. Otherwise, check with your nearest computer dealer; or to order direct, call or write:

CompuServe

P.O. Box 20212, 5000 Arlington Centre Blvd.
Columbus, Ohio 43220

1-800-848-8199
In Ohio, call 614-457-0802

An H&R Block Company

CIRCLE 7 ON READER SERVICE CARD

*Some versions of the Vidtex software do not implement all features listed.

Vidtex is a trademark of CompuServe, Incorporated. Apple is a trademark of Apple Computer, Inc. Commodore is a trademark of Commodore Business Machines. Radio Shack is a trademark of Tandy Corp.

Editor's Notes

As we see it, a special feature of our magazine is its ability to span the different machine levels—from mainframe to micro— with articles that focus on programming ideas and techniques from a conceptual point of view.

Many people have written in this month asking us why the articles in *COMPUTER LANGUAGE* seem to be mainly targeted toward one particular faction in the industry: microcomputer programming—in particular, languages and tools for the IBM and CP/M programming communities.

So I'd like to offer a challenge to readers who would like to see a greater percentage of our coverage devoted to mainframe and minicomputer programming issues. Send us your article ideas and rough drafts and we'll make ourselves open to these areas as potential topics for future issues.

This issue marks a brief editorial departure. You will find two articles that deal with hardware-related subjects, in contrast to our usual software orientation.

First, the well-known Sol Libes, former editor of *Microsystems* magazine, discusses developments in multiprocessors and operating systems. In a way not unlike his previous News and Views industry analysis column, Libes explores the evolving trend toward multiuser and multiprocessing systems using 8- and 16-bit devices.

Second, we present an article by Mike Higgins on how to create or modify the peripheral devices in any MS-DOS environment by rewriting and installing a new device driver onto the system.

Major announcement: We have an exciting project in store for the readers of *COMPUTER LANGUAGE* that should be lots of fun. We're going to build our own language from the ground up!

Namir Clement Shammas, our Cross-Xthoughts programming techniques columnist, will be our moderator throughout the development of this language, and he's now looking for people to add to his core group (see his column this month, p. 13). You can sign up with only a tentative commitment at first and see how it goes. Shammas will be using the Bulletin Board Service, CompuServe, and the U.S. mail to collect and organize the coded pieces of the eventual compiler.

One last note. A great deal of mail has come into our office recently from people who feel we should always publish in the magazine all the listings we refer to, regardless of their length.

We always try to include in the magazine the listings that are vital to the article. But sometimes, if the listing is very long or is supplemental rather than vital information, for space reasons we will put it on the *COMPUTER LANGUAGE* BBS and CompuServe and not in the magazine.

We realize this practice may pose a stumbling block to some readers who don't have an account on CompuServe or a modem and to subscribers outside the U.S. So for those readers who cannot acquire this source code electronically, we have an offer for you: just send us $2 (to cover our costs) to *COMPUTER LANGUAGE*, attention: Listings Dept., 131 Townsend St., San Francisco, Calif. 94107, and we will send you a copy of all listings that were referred to but did not appear in any particular issue of the magazine (beginning with February 1985—please specify which issue).

Thanks for your continuing support.

Craig LaGrow
Craig LaGrow
Editor

Telecommunicate to COMPUTER LANGUAGE

COMPUTER LANGUAGE has established two bulletin board systems for you to upload and download text and binary programs, as well as to leave your own electronic Letter to the Editor. All the program listings referred to in every issue of the magazine will be available here.

For those readers without access to a modem who desire a copy of program listings referred to but not printed in an issue, send $2 to *COMPUTER LANGUAGE*, attention: Listings Dept., 131 Townsend St., San Francisco, Calif. 94107. We will mail you a copy of all the listings not printed in the issue you specify (beginning with February 1985).

In addition, *COMPUTER LANGUAGE* has its own Special Interest Group on CompuServe's national data base. After calling into your local CompuServe node, simply type "GO CLM" at any prompt and you'll be in our SIG.

To access our bulletin board, set your computer or terminal to the following parameters: 8 data bits, no parity, 1 stop bit, full duplex, and either 300 or 1200 baud. The telephone number is (415) 957-9370. After your modem makes the connection, type RETURN several times, and everything else is easy.

Both systems are open 24 hours per day, 7 days per week. Due to the heavy number of callers, please do not log into the system more than one time per day. Messages left on either system will be combined the following day.

COMPUTER
LANGUAGE

EDITOR
Craig LaGrow

MANAGING EDITOR
Regina Starr Ridley

TECHNICAL EDITOR
John Halamka

EDITORIAL ASSISTANTS
Hugh Byrne, Lorilee Biernacki, John Harrison

CONTRIBUTING EDITORS
Doug Millison, Tim Parker, Namir Clement Shammas, Ken Takara,

INDUSTRY NEWS CONSULTANT
Bruce Lynch

SPECIAL PROJECTS MANAGER
Jan Dente

OPERATIONS CONSULTANT
Beatrice C. Blatteis

CIRCULATION COORDINATOR
Renato Sunico

ART DIRECTOR
Jeanne Schacht

COVER PHOTO
Dow Clement Photography

PRODUCTION ARTIST
Anne Doering

PRODUCTION
Barbara Luck, Steve Campbell, Kyle Houbolt

TECHNICAL CONSULTANT
Addison Sims

MARKETING CONSULTANT
Steve Rank

ACCOUNTING MANAGER
Lauren Kalkstein

PUBLISHER
Carl Landau

COMPUTER LANGUAGE *is published monthly by* COMPUTER LANGUAGE *Publishing Ltd., 131 Townsend St., San Francisco, CA 94107. (415) 957-9353.*

Advertising: For information on ad rates, deadlines, and placement, contact Carl Landau or Jan Dente at (415) 957-9353, or write to: COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

Editorial: Please address all letters and inquiries to: Craig LaGrow, Editor, COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

Subscriptions: Contact COMPUTER LANGUAGE, *Subscriptions Dept., 2443 Fillmore St., Suite 346, San Francisco, CA 94115. Single copy price: $2.95. Subscription prices: $24.95 per year (U.S.); $30.95 per year (Canada and Mexico). Subscription prices for outside the U.S., Canada, and Mexico: $36.95 (surface mail), $54.95 (air mail)—U.S. currency only. Please allow six weeks for new subscription service to begin.*

Postal information: Second-class postage rate is pending at San Francisco, CA and additional mailing offices.

Reprints: Copyright 1984 by COMPUTER LANGUAGE *Publishing Ltd. All rights reserved. Reproduction of material appearing in* COMPUTER LANGUAGE *is forbidden without written permission.*

Change of address: Please allow six weeks for change of address to take effect. POSTMASTER: Send change of address (Form 3579) to COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

COMPUTER LANGUAGE *is a registered trademark owned by the magazine's parent company, CL Publications. All material published in* COMPUTER LANGUAGE *is copyrighted © 1984 by CL Publications, Inc. All rights reserved.*

AZTEC

The Most Powerful C

for the IBM AT • MACINTOSH • MS DOS • CP/M-80 • ROM APPLICATIONS
IBM PC/XT • APPLE // • CP/M-86 • TRSDOS • CROSS DEVELOPMENT

Why Professionals Choose Aztec C

AZTEC C compilers generate fast, compact code. AZTEC C is a sophisticated development system with assemblers, debuggers, linkers, editors, utilities and extensive run time libraries. AZTEC C is documented in detail. AZTEC C is the most accurate and portable implementation of C for microcomputers. AZTEC C supports specialized professional needs such as cross development and ROM code development. MANX provides qualified technical support.

AZTEC C86/PRO

— for the IBM AT and PC/XT

AZTEC C86/PRO provides the power, portability, and professional features you need to develop sophisticated software for PC DOS, MS DOS AND CP/M-86 based microsystems. The system also supports the generation of ROM based software for 8088/8086, 80186, and 80286 processors. Options exist to cross develop ROM code for 65xx, 8080, 8085, and Z80 processors. Cross development systems are also available that target most micro computers. Call for information on AZTEC C86/PRO support for XENIX and TOPVIEW.

POWERFUL — AZTEC C86/PRO 3.2 outperforms Lattice 2.1 on the DHRYSTONE benchmark 2 to 1 for speed (17.8 secs vs 37.1) while using 65% less memory (5.8k vs 14k). The AZTEC C86/PRO system also compiles in 10% to 60% less time and supports fast, high volume I/O.

PORTABLE — MANX Software Systems provides **real** portability with a family of compatible AZTEC C software development systems for PC DOS, MS DOS, CP/M-86, Macintosh, CP/M-80, APPLE //+, //e, and //c (NIBBLE - 4 apple rating), TRSDOS (80-MICRO - 5 star rating), and Commodore C64 (the C64 system is only available as a cross compiler - call for details). AZTEC C86/PRO is compatible with UNIX and XENIX.

PROFESSIONAL — For professional features AZTEC C86/PRO is unparalleled.

- Full C Compiler (8088/8086 - 80186 - 80286)
- Macro Assembler for 8088/8086/80186/80206
- Linkage Editor with ROM support and overlays
- Run Time Libraries - object libraries + source DOS 1.x; DOS 2.x; DOS 3.x; screen I/O; Graphics; UNIX I/O; STRING; simulated float; 8087 support; MATH; ROM; CP/M-86
- Selection of 8088/8086, 80186, or 80286 code generation to guarantee best choice for performance and compatibility
- Utility to convert AZTEC object code or libraries to Microsoft format. (Assembly + conversion takes less than half the time as Microsoft's MASM to produce MS object)
- Large memory models and sophisticated memory management
- Support products for graphics, DB, Screen, & ...
- ROMable code + ROM support + separate code and data + INTEL Hex Converter
- Symbolic Debugger & Other Utilities
- Full Screen Editor (like Vi)
- CROSS Compilers are available to APPLE //, Macintosh, CP/M-80, TRSDOS, COMMODORE C64, and ROM based 65xx, and 8080/8085/Z80
- Detailed Documentation

AZTEC C86/PRO-AT (configured for IBM AT - options for 8088/8086)	$500
AZTEC C86/PRO-PC/XT (configured for IBM PC/XT - options for 80186/80286)	$500

AZTEC C86/BAS includes C compiler (small model only), 8086 MACRO assembler, overlay linker, UNIX, MATH, SCREEN, and GRAPHICS libraries, debugger, and editor.

AZTEC C86/BAS	$199
AZTEC C86/BAS (CP/M-86)	$199
AZTEC C86/BAS (DOS + CP/M-86)	$299
UPGRADE to AZTEC C86/PRO	$310
C-TREE Database with source	$399
C-TREE Database (object)	$149

CROSS COMPILERS

Cross Compilers for ROM, MS DOS, PC DOS, or CP/M-86 applications.

VAX - > 8086/80xxx cross	$5000
PDP-11 - > 8086/80xxx cross	$2000

Cross Compilers with PC DOS or CP/M-86 hosts are $750 for the first target and $500 for each additional target. Targets: 65xx; CP/M-80; C64; 8080/8085/Z80; Macintosh; TRSDOS; 8086/8088/80186/80286; APPLE //.

AZTEC C68K

— for the Macintosh

For power, portability, and professional features AZTEC C68K-c is the finest C software development system available for the Macintosh.

The AZTEC C68K-c system includes a 68000 macro assembler, a linkage editor, a source editor, a mouse based editor, a SHELL development environment, a library of UNIX I/O and utility routines, full access and support of the Macintosh TOOLBOX routines, debugging aides, utilities, make, diff, grep, TTY simulator with upload & download (source supplied), a RAM disk (for 512K Mac), a resource maker, and a **no royalty** license agreement. Programming examples ar included. (Over 600 pages of documentation).

AZTEC C68K-c requires a 128K Macintosh, and two disk drives (frugal developers can make do with one drive). AZTEC C68K supports the 512K Macintosh and hard disks.

AZTEC C68K-c (commercial system)	$500
AZTEC C68K-p (personal system)	$199
AZTEC C68K-p to AZTEC C68K-c upgrade	$310
Mac C-tree database	$149
Mac C-tree database with source	$399
Lisa Kit (Pascal to AZTEC C68k object converter)	$ 99

AZTEC C65

— for the APPLE //

"...The AZTEC C-system is one of the finest software packages I have seen..." NIBBLE review, July 1984.

The only commercial C development system available that runs native on the APPLE II+, IIc, and IIe, the AZTEC C65 development system includes a full floating point C compiler compatible with UNIX C and other MANX AZTEC C compilers, a 6502 relocating assembler, a linkage editor, a library utility, a SHELL development environment, a full screen editor, UNIX I/O and utility subroutines, simple graphics, and screen functions.

AZTEC C65 (Apple DOS 3.3)	$199
AZTEC C65/PRO (Apple DOS + ProDos) (call for availability)	$350

AZTEC C II/PRO

— for CP/M-80

The first member of the AZTEC C family was the CP/M-80 AZTEC C compiler. It is "the standard" compiler for development on CP/M-80. The system includes the AZTEC C II C compiler, an 8080 assembler, a linkage editor, an object librarian, a full library of UNIX I/O and utility routines, CP/M-80 run time routines, the SMALL library (creates modules less than 3K in size), the fast linker for reduced development times, the ROM library, RMAC and M80 support, library source, support for DRI's SID/ZSID symbolic debugger, and more.

AZTEC C II/PRO	$349
AZTEC CII/BAS	$199
C-TREE Database with source	$399
C-TREE Database in AZTEC object form	$149

AZTEC C80

— for TRSDOS (Radio Shack Model III & 4)

"I've had a lot of experience with different C compilers, but the Aztec C80 Compiler and Professional Development System is the best I've seen." 80-Micro, December, 1984, John B. Harrell III

This sytem has most of the features of AZTEC C II for CP/M. It is perhaps the best software development system for the Radio Shack Model III and IV.

AZTEC C80 model 3 (no floating point)	$149
AZTEC C80 model 4 (full)	$199
AZTEC C80/PRO (full for model 3 and 4)	$299

To order or for information call:

800-221-0440

(201) 530-7997 (NJ and outside U.S.A.). Or write: MANX SOFTWARE SYSTEMS, P.O. Box 55, Shrewsbury, N.J. 07701.

MANX

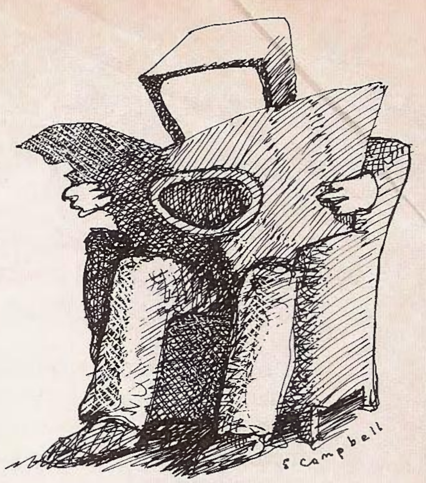
SHIPPING INFORMATION - Standard U.S. shipment is UPS ground (no fee). In the U.S. one day shipment is $20, two days is $10. Canadian shipment is $10. Two days shipment outside the U.S. is by courier and is freight collect.

TRS 80 RADIO SHACK TRS DOS is a trademark of TANDY.
APPLE DOS MACINTOSH is a trademark of APPLE.

For Technical Support
(Bug Busters) call: 201-530-6557

CIRCLE 69 ON READER SERVICE CARD

FEEDBACK



CP/M vs. MS-DOS, OS-9

Dear Editor:

In *COMPUTER LANGUAGE*'s December ComputerVisions, Bill Godbout says that (if CP/M disappears) "it wouldn't be the first time a really technically superior product went down the drain in the face of a less meritorious product."

Come on now, Bill, let's be serious! CP/M is the world's most primitive operating system. (Well, OK, there is probably something somewhere that is more primitive than CP/M.) At any rate, I regard it as the best example of being first is much more important than being any good. (The second-best example of this principle is the S-100 bus.)

Granted that MS-DOS is no great technical wonder either, but it is clearly technically superior to CP/M. At least MS-DOS (2.x) has I/O redirection, looping and testing capability in batch files, the ability to load device drivers at boot time, and hierarchical directories—none of which are available in CP/M.

I also do not share Godbout's enthusiasm for MP/M and Concurrent CP/M. Each of these adds multitasking capabilities to CP/M, but you are still basically running CP/M.

If Godbout is willing to consider other processors, Microware's OS-9 operating system leaves Concurrent and other DRI operating systems miles behind. OS-9 is a multiuser system for Motorola's 6809 processor and has been around for three or four years. It supports up to 1MB of memory and will run with just about any type and mix of peripherals. OS-9 is now available for the 68000 too, since last summer. OS-9 supports I/O redirection and other advanced features that are missing in CP/M. It is unique in allowing any command to be made memory resident (and later removed from memory if desired). Even device drivers may be loaded into memory while the system is running.

In closing, I would like to thank Bill Godbout for this opportunity to sound off about CP/M. If Viasyn (formerly CompuPro) starts making 68000-based systems, I hope Bill will consider offering OS-9 for it. CP/M-68K (another system for the 68000) is a one-cylinder engine trying to power a Ferrari.

Jim Howell
San Jose, Calif.

Not a number cruncher

Dear Editor:

Tim Parker writes in the December Public Domain Software Review that APL, while a "number cruncher at heart," is not a general purpose language.

He is correct as to its crunching but gives the wrong impression as to its generality. The 80 or so primitive (in the primal sense) functions of the language and the 120 or so system functions cover a very broad range of needs in business, finance, engineering, science, actuarial work, planning—you name it.

Two comments regarding this: the APL*PLUS/PC package from STSC Inc. has just been ranked, I am told, 10th among PC software; you can scan the features that other languages have and find that just about ALL are—and most have been for years— part of APL. The news is that APL became broadly available, in a quality system, for the PC only two years ago. Before that, we were limited to minis or mainframes, except for some small-scale attempts on personal computers and an abortive attempt by IBM on its 5100.

As an example of language comparison, consider Table 1 in Steve Heller's Exotic Language of the Month Club column in the same December issue: he rates OMNI at 55. APL would have to rate at least 60 on just those criteria he has picked. To mention only a few, APL handles: arrays of up to 63 dimensions and 65K bytes; integer and floating point numbers automatically as appropriate, localized variables at any level of nesting functions; and, being interpretive, APL has no source vs. object code, only very dense code by virtue of its symbols, which is not an insignificant advantage on micros. The Sieve of Eratosthenes benchmark takes 125 bytes of code and runs in less than 50 sec on a Compaq.

APL is probably the highest level, general purpose language available today. Its power can perhaps be illustrated by a sorting example—suggested by the amount of space devoted to Quicksort in both the November and December issues. I made up a small example with cities and their population.

This shows the typical way in which we work on arrays in APL. And it illustrates that, as APL users—not systems programmers—no one has worried about sorting algorithms in a decade or more. I just sorted 10,000 random integers in under 5 sec and 8,000 floating-point numbers (just under the limit of object size) in 14 sec. Rather, we have been more concerned with extending the use of existing functions, for instance the sorting function (which consists really of finding a reordering sequence and applying it to an array by indexing) to more complex procedures such as melding.

The Sieve of Eratosthenes benchmark code and the sorting and melding examples are available for downloading on the *COMPUTER LANGUAGE* Bulletin Board Service (415 957-9370) and CompuServe account (type "GO CLM").

If the aim of the game for users of micros—as opposed to those who build the micro systems themselves—is to develop applications quickly and efficiently, APL is generally hard to beat and deserves careful comparison against other languages.

George Brigham
President, The APL Group Inc.
New Canaan, Conn.

Columnist Tim Parker responds: Unfortunately, a number of people misinterpreted the statement I made in my December column. I maintain that APL is a number cruncher par excellence and is probably unsurpassed in that role. I did not say that it could not do anything else! I merely pointed out its primary attribute.

I am well aware that APL can handle text and string manipulation but have to wonder whether I would choose the lan-

LOMAS DATA PRODUCTS INVITES YOU TO:

SHARE THE THUNDER.

The S100-PC-TM offers the following standard features:

- ☐ High performance THUNDER186 8Mhz 80186 processor
- ☐ 512K bytes of RAM (expandable to 1Mbyte)
- ☐ 4 serial ports to support up to four users
- ☐ 3 Centronics compatible parallel ports
- ☐ Concurrent DOS operating system allows execution of both CP/M-86 and MS-DOS (PC-DOS) programs
- ☐ 5¼″ IBM-PC compatible floppy drive
- ☐ 40 Mbyte high performance Winchester drive
- ☐ Attractive 10 slot desktop enclosure

In addition, a number of options are available including: larger Winchester drives, more user ports, 80286 processor, graphics support and additional operating systems (MS-DOS and CP/M-86).

S100 BUS boards products & support for the system integrator . . .

All of LDP boards are fully tested to exacting standards and carry a one year warranty. We specialize in 16-bit products & support the major operating systems for 16-bit processors: CP/M-86*, CONCURRENT CP/M-86*, and MS-DOS (PC-DOS).

■ **THUNDER186 — THE ONLY COMPLETE S100 BUS, 16 BIT SINGLE BOARD COMPUTER AVAILABLE TODAY.**
Concurrent CP/M-86, which in addition to running CP/M-86 programs, runs MS-DOS programs. Comes complete, ready to plug into an enclosure and run. 256K bytes of RAM only . . . . **PRICE $1595.00**

■ **LIGHTNING ONE***8086/8088 CPU**
8086 or 8088, with 8087 and 8089 coprocessors. Up to 10 MHZ operation . . . . **PRICES start at $425.00**

■ **HAZITALL SYSTEM SUPPORT BOARD**
2 serial, 2 parallel ports, battery protected clock calendar. Hard disk controller host interface . . . . . . **PRICE $325.00**

■ **LDP 128/256K DYNAMIC RAM**
Advanced dynamic RAM with LSI controller for failsafe operation, parity **PRICE** 128K—**$495.00**, 256K—**$795.00**

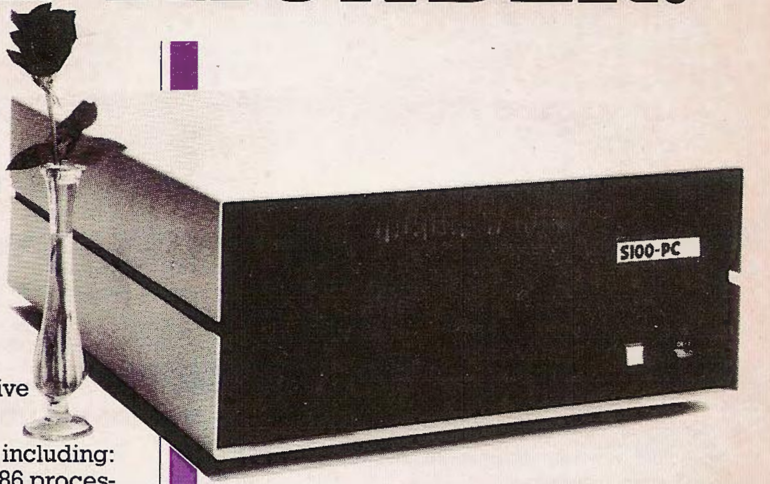
■ **RAM67 HIGH PERFORMANCE STATIC RAM**
High speed (100ns) low power CMOS static RAM. 128K bytes, extended addressing . . . . **PRICE $995.00**

■ **LDP72 FLOPPY DISK CONTROLLER**
Single/double density, single/double sided disks, both 8″ and 5¼″ inch drives simultaneously . . . . . **PRICE $275.00**

■ **LIGHTNING 286—80286 CPU BOARD**
Offers 4 times the performance of a 5MHZ 8086 CPU while maintaining software compatibility . . . . . **PRICE $1395.00**

■ **OCTAPORT 8 PORT SERIAL BOARD**
0 to 19200 baud operation real time clock interrupt. Ideal for multi-user systems such as MP/M-86* **PRICE $395.00**

S100-PC-TM is a trademark of Lomas Data Products, Inc.
**CP/M-86, MP/M-86 and CONCURRENT CP/M-86 are trademarks of Digital Research. **MS-DOS is trademark of Microsoft.*
****Lightning One is trademark of Lomas Data Products, Inc.*



S100-PC-TM: The LDP Multi-user S100 Bus System offers high performance at a "low" price . . . plus, "our" system is expandable and upgradeable!

PRICE
An unbelievable **$6995.00**

Call today!

LDP

LOMAS DATA PRODUCTS, INC.
66 Hopkinton road, Westboro, MA 01581
Tel: (617) 366-6434 ☐ Telex: 4996272

Dealer inquiries invited.

For orders outside the U.S., contact our exclusive dealers: ☐ **Australia** —
LAMRON PTY. LTD., (02) 85-6228
☐ **Malaysia** — EXA COMPUTER (M)
SENDIRIAN BERHAD, 795284

CIRCLE 54 ON READER SERVICE CARD

guage for a string-oriented application on that basis. It remains choice among the language arsenal for mathematical applications though.

Having programmed with APL for several years on both mainframes and micros, it remains one of my favorite languages— not only because of its power but also because of its versatility and speed. The code even looks impressive to non-programmers!

COBOL dispute

Dear Editor:

I cannot let Robert Wagner's letter in the January 1985 issue of *COMPUTER LANGUAGE* pass unchallenged. I am very surprised that a professional in computer science has so unclear an understanding of the features of computer languages or of compiler technology. In fact, the letter serves only as a plug for a particular IBM PC COBOL compiler and an undocumented plug at that.

Compilers are programs, just like any other applications program. As programs, they do certain things (character input, string manipulation, data structure maintenance, object output) unique to their applications domain. Thus, a language that allows a programmer to conveniently express these operations is one which is suitable for writing compilers. It is nonsense to claim that a compiler for a particular language is always best written in that language. For example, how would one write a compiler for APT, a machine-tool manipulation language, in APT?

Wagner further claims, on the basis of the performance of the Realia COBOL compiler, that COBOL is "inherently faster than the rest." To conclude that a language is inherently fast or slow on the basis of one outstanding (or outstandingly bad) implementation is a dangerous assumption. Interpreted BASIC on a IBM 3083 (with one user) considerably outperforms the small C compiler running under CP/M; does this mean that Interpreted BASIC is inherently faster than C on the same machine?

Wagner's final claim—that his PC with the Realia compiler beats an IBM system running on 3370 disc packs—is hardly as surprising as it seems. At work I use a VAX 11/780 and a high-performance JOVIAL compiler. Programs, in general, run somewhat slower than they do written in Turbo Pascal on my Kaypro 10. Why? Because there are 10, 20, even 25 people using the VAX simultaneously; fortunately, my Kaypro is laboring under no such limitation.

I am glad that Wagner has found such an exceptional compiler in the Realia COBOL implementation. I would be very interested in some controlled, documented benchmarks. But making claims about a language based on a particular compiler are ill-founded.

Christopher Pettus

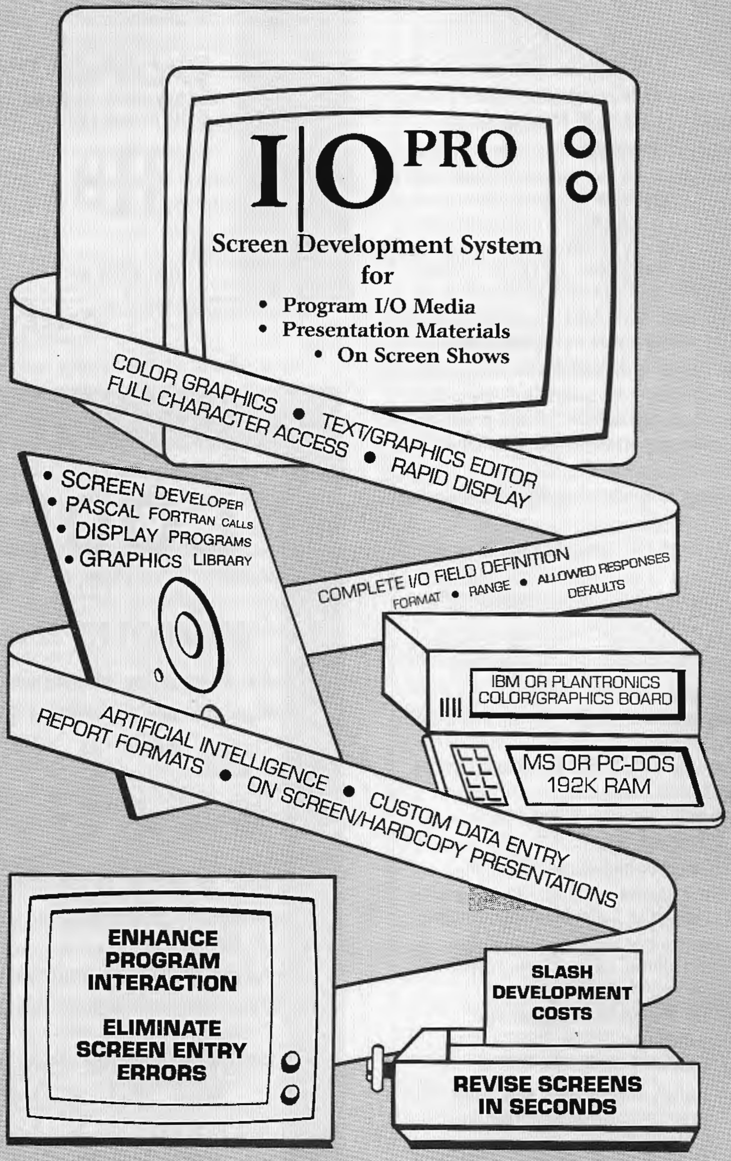
C to Assembly

The first two source code listings in "C to Assembly Interface" in the February issue had a portion of the code duplicated in the beginning of the listings and a few other problems.

In Listing 1, the TITLE line was replaced with a chunk of code from the middle of the listing. In addition, "POP DX" should have followed "POP CX" in the otherwise correct portion of the listing. To correct this listing, delete the first 13 lines and replace them with the line "TITLE DSPIO - SUBROUTINE TO PERFORM DIRECT SCREEN I/O". Then insert the line "POP DX" after the line "POP CX".

In Listing 2, delete the first 14 lines and the characters on the fifteenth line preceeding "define PAGE O".—Ed.

I/O A BORE? NOT ANY MORE!


I|O PRO
Screen Development System for
• Program I/O Media
• Presentation Materials
• On Screen Shows
COLOR GRAPHICS • TEXT/GRAPHICS EDITOR
FULL CHARACTER ACCESS • RAPID DISPLAY
• SCREEN DEVELOPER
• PASCAL FORTRAN CALLS
• DISPLAY PROGRAMS
• GRAPHICS LIBRARY
COMPLETE I/O FIELD DEFINITION
FORMAT • RANGE • ALLOWED RESPONSES DEFAULTS
IBM OR PLANTRONICS COLOR/GRAPHICS BOARD
MS OR PC-DOS 192K RAM
ARTIFICIAL INTELLIGENCE • CUSTOM DATA ENTRY
REPORT FORMATS • ON SCREEN/HARDCOPY PRESENTATIONS
ENHANCE PROGRAM INTERACTION
ELIMINATE SCREEN ENTRY ERRORS
SLASH DEVELOPMENT COSTS
REVISE SCREENS IN SECONDS

M|E|F Environmental, Inc.
P.O. Box 26537
Austin, TX 78755 (512) 251-5543

$450 : OEM Pricing and Licenses Available
Demonstration Diskette $10
applicable to purchase

CIRCLE 15 ON READER SERVICE CARD

F77L LCS-FORTRAN FOR IBM/PC CHALLENGES SOFTWARE GOLIATHS

UTILITIES
COBOL
C
BASIC
FORTRAN
PASCAL
WORD PROCESSING
GAMES
GRAPHICS
DATA MANAGEMENT
SPREADSHEET
F77L
LCS

When you have the superior product you can take on (and beat!) the big guys.

Up until now FORTRAN programmers have had to get by with ponderous and incomplete Language Systems that never fully meet their needs or expectations. With F77L Lahey Computer Systems resolves this situation. F77L is a complete ANSI 77 Standard FORTRAN Language System with fast compile speed and productivity-enhancing diagnostics.

If you need a FORTRAN Language System, don't go to a Goliath who just happens to carry a FORTRAN product: call LCS, the company with the superior product.

Here are a few of the many reasons to buy F77L:

- Full FORTRAN 77 Language with popular extensions.
- Fast, compiles 100s of statements/minute (PC/XT).
- Numerous, specific English language diagnostics displayed during compilation.
- Command Line compiler options.
- Execution error traceback: program unit & line number.
- Optional protection for constants, bonds, interfaces.
- Standard or Free Format source files.
- Lattice C compatibility.
- Optimized code or high-speed execution.
- Easy to use manual includes appendices on interfaces to Lattice C and Assembly language.

If you are tired of betting on the software Goliaths, and losing, call LCS the FORTRAN SPECIALIST—ask for David.

$477 for complete package.
Requires: 256K/8087
FORTRAN IS OUR FORTE

Lahey Computer Systems, Inc.
31244 Palos Verdes Drive West,
Suite #243
Rancho Palos Verdes, CA 90274
213/541-1200

Serving the FORTRAN Community since 1969.
IBM is a trademark of IBM Corporation
Lattice C is a trademark of Lattice, Inc.

CIRCLE 22 ON READER SERVICE CARD

ConIX™

UNIX™ Technology for CP/M™

ConIX can provide any 48K+ CP/M-80 compatible system with many advanced capabilities of UNIX. You'll be amazed at what your 8-bit micro can do now! ConIX features include:

I/O Redirection and Pipes (uses memory or disk), multiple commands per line, full upper/lower case and argument processing, Auto Screen Paging, Programmable Function Keys, improved User Area Directory manipulation, Command and Extension (Overlay) Path Searching, "Virtual" disk system, 8Mb Print Spooler, extensive preprocessed "Shell" command programming language, 300+ variables, over 100 built-in commands, Math Package, 22 new BDOS SysCalls, Archiver (compacts files for disk space savings of over 50%), On-Line Manual System, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs quickly and easily without any system modifications.

The ConIX Operating System
List Price: $165

Price includes Instructional Manual, 8" SSSD disk, and free support. 5¼" format conversions available. To order, contact your local dealer, or buy direct and add shipping: $4.50 UPS, $10 Canada, $25 overseas. COD $2 extra (USA only). NY State residents add sales tax.

Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786

Dealer inquiries invited!

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

CIRCLE 12 ON READER SERVICE CARD

CODE SIFTER

THE EASY TO USE
SYMBOLIC EXECUTION PROFILER

No knowledge of assembly language is needed.

Easy to use — become productive in minutes.

Online HELP at each menu complements the user manual.

Define 32 ranges in USER/DOS/BIOS with a single keystroke or enter them manually if you desire.

Use the looping feature to isolate the busy areas of your program automatically.

Symbolic statistical output can be sent to any device or file.

Adjustable sampling rate for increased accuracy.

Free demonstration diskette available. (requires deposit)

David Smith Software
Box 25A R.D.#3
Oxford, N.Y. 13830
(607) 843–6209

Requires IBM–PC
128K DOS 2.X

$119

IBM is a trademark of International Business Machines Corp.

CIRCLE 36 ON READER SERVICE CARD

INDUSTRY INSIGHT

By Bruce Lynch

Low-cost and/or introductory language implementations are quickly becoming an important trend in the industry. The concept of quality, low-cost languages and tools may have a dramatic effect on how the competition will stack up in the next few years.

For example, for $95 one can acquire such implementations as: an Ada subset by RR Software, Pocket APL by STSC, Introducing-C by Computer Innovations, Professional BASIC by Morgan Computing, and a Pascal by Software Building Blocks. For $90, a Modula-2 for MS-DOS or the Macintosh can be bought from Modula Corp., and a Forth from Laboratory Microsystems is $100.

Joining Borland International's Pascal at $50 is an intriguing MTBASIC. Plus, for some time now there have been low-priced implementations from Ellis Computing of FORTRAN, COBOL and Pascal. The ubiquitous Turbo Pascal seems to have started the trend. Other low-cost implementations exist and yet more are planned soon from Manx, Lifeboat, and others.

Low-cost implementations of other languages will help feed the desire of many who know BASIC and want to try another language. Meanwhile, numerous companies are working at adding capabilities to BASIC to make it more attractive.

Morgan's lowering the price of Professional BASIC to $95 is a step that is more likely to help in the struggle for attention between improved BASICs and alternative languages than all of the marketing of the other products combined. When a person is not sure if he or she will end up using a software product after experimenting with it, there is a cost threshold above which almost all such potential users are lost. Demo versions of Morgan's BASIC for $5 and Summit Software's Better BASIC for $10 are an effective means of helping a potential user to be sure of the long-term value of the product before risking much.

Another trend strongly pushed along by Turbo Pascal is integration of an editor with a compiler. Bellesoft's ES/P seems to have been the first for micros with its syntax-sensitive ES/P editors for Pascal and C. FirstTime by Spruce is a product similar in design. Others that are also tightly tied to the syntax, like Fast C by Lifeboat, are close to becoming available.

Others editors are offering optional and reconfigurable compiler integration. Many programmers prefer the option of turning off certain features and resetting the defaults for others. Facilities for programmers to control separately often include: running a compiler without leaving the editor, error-to-error movement, auto-indent, use of common language constructions, and others. PMATE users have written their own such macros and Phoenix Software is rumored to be releasing a special version for C soon. Other products with such capabilities are expected soon.

Numerous categories of products to improve programmer productivity continue to appear while others languish in labs. Program generators designed for programmers and others designed for end users are likely to become viable within the next couple of years.

The large number of products containing libraries of prewritten programs for graphics, screen generation, file management or other common program characteristics are another part of this trend. Some of them have special editors to make the process more intuitive and easier for laying out what the programs are supposed to do.

Special languages were previously a lost cause because relatively few people within a specialized group had access to computer terminals. Now, with the number of micros installed, languages for data management/reporting (dBASE) for spreadsheet reporting and manipulation (Lotus 1-2-3), are likely to be joined by other highly accepted languages. Possible application areas include engineering calculations (TK! Solver is an attempt), statistics, graphics, communications, and accounting transactions (Sensible Solution is an attempt).

With the addition of natural language features, specialized languages will become viable for people who have less time to invest in learning. CLOUT and SALVO are examples of products designed to offer a lower learning curve.

Only 18 months ago, C was targeted only at systems programmers. By late 1983 support tools and changes to compilers available for microcomputers began to make C a language appropriate for use by application programmers. Support libraries became available to cover a wide variety of miscellaneous needs, like clearing the screen, accessing MS-DOS support services, fundamental screen and graphics, etc. They make application programming realistic for those who don't know how to write such routines themselves.

By early 1984 books designed to help someone with minimal or no programming background to learn to program in C began to appear. Specialized libraries that cropped up in the summer of 1983 were quickly joined by competitors.

Now the market is virtually flooded with both specialized libraries and miscellaneous libraries for use by C programmers. For instance, there are now at least five new file-management libraries that have become available for C programming during the last several months.

C is becoming usable by people with less and less programming experience. The next step in this trend is beginning to show some evidence through product announcements.

Implementations of C are now available that are as interactive and integrated as Microsoft BASIC for MS-DOS. Examples of these products include Instant C by Rational Systems and Introducing C by Computer Innovations. You should expect to see a Turbo C from Borland or some other company by late first quarter of 1985. (101 C programs for the Dentist is not far behind—here's the introduction of interpreters.) If the trend can stay in force, one should expect to see an implementation of C bundled in with high-end home computers by early 1986.

Two substantial forces threaten the continuation of the popular C trend happening. Those two forces are intrinsic characteristics of C that make it difficult for a beginner to use. They also potentially strengthen the status of competing languages that may be better suited to the needs C would otherwise address.

The lack of strong type checking and other features which help prevent a programmer from hurting him or herself are characteristics fundamental for what C was originally intended to accomplish. Because of the power of the current trend, compiler writers and others are attempting to introduce changes to their products that will focus on these issues.

Modula-2 seems to have the potential to address the market toward which C is currently moving. Continued efforts to implement versions of BASIC which better fulfill the needs of the casual programmer also represent an external threat to everyman C.

The ANSI standards meetings on the future of C are led by practical and pragmatic people. Characteristics likely to be added will be helpful to programmers without stripping C of its flexibility and power.

Specialized front ends for C will likely appear in the next six months. Front ends will have numerous different characteristics. Products with front ends will overlap each other a fair amount. The front ends will help with source-level debugging, a major need. Front ends are also used for dynamic checking and for static support, which keeps a programmer from doing something dangerous.

Large development shops outside the microcomputer industry have been somewhat reluctant to adopt C as their primary, large-project language. Books about programming standards and front-end processors should help to serve the need for standardization, making it more viable for these people.

Before I incur the wrath of compiler or interpreter writers partisan to other languages, remember that what has been said here has been focused only on C. When one considers the trends pertinent to Pascal, BASIC, Assembler, Forth, Modula-2, Ada, FORTRAN, COBOL, etc., the picture is much more confusing.

I do not expect C to become the dominant language in the lives of every category of programmers. It may not even have a major impact as the language for casual programmers. Remembering that several hundred thousand computers are being sold each month with a copy of BASIC, C will have a tough time selling more than even 20,000 copies per month.

By the way, my guess is that approximately 2,500 to 4,000 copies of C are being sold for use on single-user microcomputers per month now. ■

TOTAL CONTROL:

FORTH: FOR Z-80®, 8086, 68000, and IBM® PC

Complies with the New 83-Standard

GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS
DATA ACQUISITION • PROCESS CONTROL

- ***FORTH*** programs are instantly portable across the four most popular microprocessors.
- ***FORTH*** is interactive and conversational, but 20 times faster than BASIC.
- ***FORTH*** programs are highly structured, modular, easy to maintain.
- ***FORTH*** affords direct control over all interrupts, memory locations, and i/o ports.
- ***FORTH*** allows full access to DOS files and functions.
- ***FORTH*** application programs can be compiled into turnkey COM files and distributed with no license fee.
- ***FORTH*** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth + and PC/GEN, Laboratory Microsystems, Inc.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities and 200 page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, $100.00; **8080 FORTH** for CP/M 2.2 or MP/M II, $100.00; **8086 FORTH** for CP/M-86 or MS-DOS, $100.00; **PC/FORTH** for PC-DOS, CP/M-86, or CCPM, $100.00; **68000 FORTH** for CP/M-68K, $250.00.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

PC FORTH +	$250.00
8086 FORTH + for CP/M-86 or MS-DOS	$250.00
68000 FORTH + for CP/M-68K	$400.00

Extension Packages available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure.*

VISA

LMI

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to ***(213) 306-7412***

MasterCard

CIRCLE 35 ON READER SERVICE CARD

UNPARALLELED
PERFORMANCE
and
PORTABILITY
in an
ISAM PACKAGE
at an
UNBEATABLE
PRICE

The company that introduced micros to B+ Trees in 1979 and created ACCESS MANAGER™ for Digital Research, now redefines the market for high performance, B+ Tree based file handlers. With c-tree™ you get:

- complete C source code written to K&R standards of portability
- high level, multi-key ISAM routines and low level B+ Tree functions
- routines that work with single-user and network systems
- no royalties on application programs

$395 COMPLETE

Specify format:
5¼" PC-DOS 3½" Mac
8" CP/M® 8" RT-11

for VISA, MC or COD orders, call
1-314-445-6833

c-tree™
BY FAIRCOM

2606 Johnson Drive
Columbia MO 65203

Access Manager and CP/M are trademarks of Digital Research, Inc.
c-tree and the circular disc logo are trademarks of FairCom

© 1984 FairCom

CIRCLE 29 ON READER SERVICE CARD

CROSSXTHOUGHTS

Mathematical parsers and interpreters

By Namir Clement Shammas

I have a special challenge to offer in the form of a question. Wouldn't it be great fun for all you language wizards out there to combine your talents to create a language we could call our own? We could even set up a special club both on CompuServe and the Bulletin Board System for this new language's development!

Why another language? Same reason for having another Olympic game: the challenge. The language can be oriented toward statistics, chemistry, or any other specialty. We could selectively create a language that takes only the most important ideas from other languages—like portability, structured design, and modularity.

On this theme, allow me to throw out an idea. Perhaps a way of doing this collective programming project would be to create a new language that is a subset of the Ada language. And perhaps we could call it Little Ada (LADA).

Write in and tell me if you think this is a good or bad choice or how you would go about structuring this *COMPUTER LANGUAGE* reader development team. We're just at the idea stage now, so tell me what you think!

Our subject in this issue is an interactive parser and interpreter for mathematical expressions. This includes the cases where a program user has the ability to interactively alter any number of equations for as many times desired. Electronic spreadsheets use the latter feature.

Another example of use is statistical regression programs that allow the regression model to be altered dynamically without recompiling the entire program. Programs that plot mathematical functions, obtained from the user interactively, also fall in this category. Thus we need to develop a parser/interpreter integrated in an application that handles the verification and evaluation of the assignments and expressions.

The process of developing the parser/interpreter involves a lot of variations and choices that define the features of the system.

The first variation is the type of expressions. While postfix expressions, also known as reverse Polish notation (RPN), are evaluated in a straightforward fashion, few of us, if any, write equations in RPN! Infix expressions are popular, but they are not evaluated smoothly because the operations and functions have different priorities, something RPN has avoided altogether.

It becomes clear that the system we want will consist of two parts. The first, the parser, converts from infix to postfix notation and performs error checking. An equivalent RPN expression is produced. Its evaluation is carried out by the second part, the interpreter. This defines the general scheme of things.

Considering the expression parsers, there can be several types:

Nonrecursive. The expression is scanned from left to right, one character at a time. While scanning, the program looks for operators, parentheses and variable names. A number of temporary stacks are used to handle encountering operations with superior priorities, functions, and subexpressions.

Recursive. This kind is implemented in languages that allow for recursion, such as C, Pascal and Modula-2. It breaks down the expression into similar subexpressions and finally produces the equivalent RPN notation for each.

Interactive recursive. Using a mini-syntax-oriented editor, a user is guided step by step in entering a correct expression. This type has an easier time tracking an error and reporting it instantaneously. The use of recursion allows for the entry of subexpressions.

Another variation that defines the syntax of the expression and the features of the parser is the representation of variables. This can range from simple to complex as dictated by the nature of the application. We have the following types:

Single arrays or matrices. In some cases the data is processed as one matrix or a set of vectors read one at a time. So one variable name is sufficient as long as it is followed by a one- or two-dimensional array index. Storing the data internally is easy since the identifier name is ignored. A parallel array or matrix is used for data storage.

Multiple distinct names for variables, arrays and matrices. This requires more elaborate schemes to identify the variables and manage its data. An internal identifier table must be constructed. Hashing techniques are most likely needed to quickly verify the existence of an identifier and locate it. When arrays and matrices having different names are used, the identifier name and the indices serve as pointers to the actual data storage areas.

The operations permitted are another factor in determining the syntax of an expression in an application. One can implement the basic operations only. Another application may require the use of logical operators.

The use of mathematical functions adds another variation in defining the tasks of the parser/interpreter system. You may decide that every function must begin with the same special symbol, such as @. This can help the parser distinguish between function and variable names. The number of functions allowed can also vary from one application to another.

Having discussed the variations that define the expression syntax and the parser's action, I will proceed with an example to show more details in defining the equation parser and RPN interpreter. It is written in the Programmer's Pseudo Listing (PPL), loaded on the BBS in a file called NILE. (More information on PPL can be found in February's CrossX-thoughts column.) The first step is to define the system. It will consist of:

- Interactive recursive parser
- RPN expressions are produced with special symbols to speed interpretation
- Variables are implemented as the single array *X01* through *X<nn>* where *nn* is defined by the software author
- The basic four operations and negations are allowed
- Logical operators
- Functions are allowed, though treated generally. Function names need not be preceded by any special symbol.

Listing 1 shows the pseudocode of the

GROWING OLD?
waiting
for C programs to
compile and link..?

Use **C-terp**
the **complete** C interpreter

This is the product you've been waiting (and waiting) for!

Increase your productivity and avoid agonizing waits. Get instant feedback of your C programs for debugging and rapid prototyping. Then use your compiler for what it does best...compiling efficient code ...slowly.

C-terp Features
- Full K&R C (no compromises)
- Complete built-in screen editor-- no half-way house, this editor has everything you need such as multi-files, inter-file move and copy, global searching, auto-indent, tab control, and much more.
- Fast--Linking and semi-compilation are breath-takingly fast. (From edit to run completion in a fraction of a second for small programs.)
- Convenient--Compiling and running are only a key-stroke or two away. Errors direct you back to the editor with the cursor set to the trouble spot.
- Compiler Compatible--You can access functions and externals compiled with C86 or Lattice C or assembly language. Utilize your existing libraries unchanged!
- Complete Multiple Module Support-- Instant global searches, auto-compile everything that's changed, etc.
- Many more features including batch mode and symbolic debugging.
- Runs on IBM PC, DOS 2.x, 192K
- **Price: $300.00 (Demo $45.00) MC, VISA**

Price of demo includes documentation and shipping within U.S. PA residents add 6% sales tax. Specify C86 or Lattice version.

GIMPEL SOFTWARE
3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

*Trademarks: C86 *(Computer Innovations)*, Lattice *(Lattice Inc.)*, IBM *(IBM Corp.)*, C-terp *(Gimpel Software)*

CIRCLE 50 ON READER SERVICE CARD

```
MODULE Main
-- There are two "windows".  Windowl is were the expression is
-- being entered.  Window2 is for messages and messages and
-- menus.
DISPLAY on windowl, "X"
INITIALIZE: None.
LOOP <One>
BEGIN

      DISPLAY on window2 "Enter index for assigned variable"
      INPUT Index
      IF Index <= Allowed_limits THEN EXIT <One> END IF
END LOOP <One>
TERMINATE: None  -- Nothing pending
DISPLAY on windowl, Index;" :="
Function_flag = FALSE
RPN_Expression = {Get_Expression}
IF Function_flag THEN
          Append(RPN_Expression,Function_Name)
          Function_flag = FALSE

END IF
Choice from Menu(Skip, '=', '<', '>', '<>', '>=')
IF Choice <> Skip THEN
          Append(RPN_Expression,{Get_Expression})
          IF Function_flag THEN
               Append(RPN_Expression,Function_Name)
               Function_flag = FALSE
          END IF
          Append(RPN_Expression,Choice)

END IF
Append(RPN_Expression,ADDRESS(Index),STORE)
END Main

IMPLICITLY RECURSIVE FUNCTION Get_Expression
Choice from Menu(Skip, '+', '-')
Build_Expression = {Get_Term}
IF Choice = '-' THEN Append(Build_Expression, 'CHS') END IF
Choice from Menu(Skip, '+', '-', 'OR')
INITIALIZE: None
LOOP <Two>
BEGIN IF Choice = Skip THEN EXIT <Two> END IF
     Append(Build_Expression{Get_Term},Choice)
     Choice from Menu(Skip, '+', '-', 'OR')
END LOOP <Two>
TERMINATE:  Get_Expression = Build_Expression
END Get_Expression

FUNCTION Get_Term
Build_Term = {Get_Factor}
Choice from Menu(Skip, "*', '/', 'DIV', 'MOD', 'AND')
INITIALIZE:
LOOP <Three>
BEGIN IF Choice = Skip THEN EXIT <Three> END IF
     Append(Build_Term,{Get_Factor},Choice)
     Choice from Menu(Skip, '+', '-', 'OR')
END LOOP <Three>
TERMINATE: Get_Term = Build_Term
END Get_Term
```

Listing 1 *(Continued on following page).*

interactive parser. The approach is a structured one. The *Menu* and *Append* procedures are really generic. Notice that the number of arguments varies throughout the listing. They represent a collection of functions that handle the variables with the levels of an expression, namely terms and factors. I will not list any pseudocode for *Append* and *Menu* since their functions are easily written.

The main program calls on *Get_Expression* to build the equation. The *Get_Expression* function calls on *Get_Term* which in turn calls on *Get_Factor*. The latter may call on a number of functions to enter either a numeric constant, a variable, a function, or an expression. (That is why we call the parser recursive.)

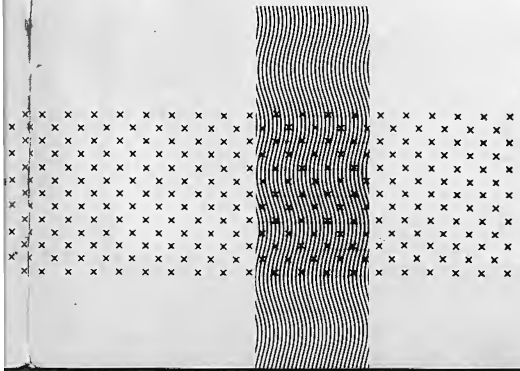
Only the pseudocode for entering a number is shown; the rest is omitted because it is fairly straightforward, and we have limited space. It is worthwhile to point out that these routines are responsible for error checking and recovery.

The listing for the function *Get_Constant* is interesting. There are two loops. The outer is repeated until a syntactically correct number is entered. The inner loop cooperates with a *CASE-OF* construct to verify the entered number. The variable *Switch* is used to point out the different examination phases.

Each *CASE* option handles a certain stage of the verification. The loop allows for repeating stages, specifically those scanning for a sequence of digits. As each stage is completed, the value of *Switch* increases incrementally. Since a potential syntax error can occur in many places, writing the code with nested *IF-THEN-ELSE* constructs does not yield a readable listing, at least not for me!

While an infix expression can be converted to an equivalent postfix (RPN) expression, as shown in Figure 1, spaces separate the components of the RPN. This allows for the interpreter to process the expression properly.

There still remains one small problem, independent of the method used to obtain the RPN expression. The speed of evaluation is important, and the interpreter may still waste time trying to discern the text at hand.



```
FUNCTION Get_Factor
Choice from Menu(Constant,Variable,Function,Expression)
CASE Choice OF
     WHEN Constant   => Get_Constant
     WHEN Variable   => Get_Variable
     WHEN Function   => Get_Function_Name from Menu(function list)
                        Function_flag = TRUE
                        Get_Expression
     WHEN Expression => Get_Expression
END CASE
END Get_Factor

FUNCTION Get_Constant
-- Number is defined by the BNF notation below
-- [+|-] digit [{digit}]"."[{digit}][ [+|-]"E"|"e" digit [digit]]
INTIALIZE: OK = TRUE
LOOP <OUT>
  DISPLAY in window2,"Enter number"; INPUT String
   INTIALIZE: Ptr = 1; Switch = 1
   LOOP <IN>
    CASE Switch OF
     -- Check first character
     WHEN 1 => IF String[Ptr] IN [+,-,digit]
                    THEN Ptr += 1; Switch = 2
                    ELSE OK = FALSE
     -- Check decimal or possible repetition of digits
     WHEN 2 => IF String[Ptr] IN [.,digit]
                    THEN Ptr += 1
                       IF  String[Ptr] = '.'
                              THEN Switch = 3 END IF
                    ELSE OK = FALSE
     -- Check digits after decimal, exponent or End Of Line
     WHEN 3 => IF String[Ptr] IN [digit,E,e,EOL]
                    THEN  CASE String[Ptr] OF
                            WHEN EOL => Switch = 10 -- Exit
                            WHEN E,e => Switch = 4; Ptr += 1
                            WHEN digit => Ptr += 1
                          END CASE
                    ELSE OK = FALSE
     -- Check that exponent is correct
     WHEN 4 => IF String[Ptr] IN [+,-,digit]
                    THEN IF String[Ptr] = digit
                              THEN Ptr += 1; Switch = 5
                              ELSE Ptr += 1;
                                   IF String[Ptr] = digit
                                        THEN Ptr += 1; Switch = 5
                                        ELSE OK = FALSE
                                   END IF
                         END IF
                    ELSE OK = FALSE
               END IF
     -- Check that number terminates properly
     WHEN 5 => IF String[Ptr] IN [d,EOL] THEN Switch = 10
                                         ELSE OK = FALSE
                END IF
   END CASE
   IF (Switch = 10) OR (NOT OK) THEN EXIT_LOOP "IN" END IF
 END LOOP <IN>
 TERMINATE: None
 IF OK THEN EXIT <OUT> END IF
END LOOP <OUT>
-- Perform the assignment below, now that the number has been
-- verified.
TERMINATE: Get_Constant = String
END Get_Constant

END Main
END PPL
```

Listing 1 *(Continued from preceding page).*

The solution lies in imbedding special symbols right after the spaces that point out the nature of the component to be dealt with. Figure 1 shows an RPN expression with some suggested symbols. Listing 2 shows the interpreter, assuming that the above symbols were already inserted by the *Append* function, for speeding up the evaluation process.

And now, our minicontest! Send your code for a nonrecursive, recursive, or interactive recursive parser and interpreter, written in any of the following langauges: BASIC, C, Pascal, Modula-2 and Ada.

The author of the best code (including readability, modularity, speed, and features) will be awarded a copy of BDS C compiler. The deadline for entries is May 3, 1985. Send your entries to *COMPUTER LANGUAGE* in a letter or via the BBS or CompuServe (the last two methods are preferred). The results will be announced in the June issue.

If you are interested in further discussions on mathematical equations parsers, send me a note c/o *COMPUTER LANGUAGE* to put your name in a SIG-MEP list. There is a lot more material that is not published due to space limitations.

In the next issue we will be talking about fast searching techniques (RAM based or disk searches) that employ hashing.

```
Infix :   X1 := Sqrt(X2 + 2.5 * X3) / ((X2 + 1) * X3 + 10)

Postfix:  X2  2.5 X3 * + Sqrt X2 1 + X3 * 10 + / ADDRESS(X1) :=

Postfix with suggested special symbols to aid the interpreter.

     !X2 #2.5 !X3 |* |+ @Sqrt !X2 #1 |+ !X3 |* #10 |+ |/ =X1

where ! signals a variable
      # signals a number
      | signals an operation
      @ signals a function
      = signals an assignment
```

Figure 1.

```
MODULE Interpreter

-- Table for symbols inserted by parser' APPEND generic functions
-- ! signals a variable
-- # signals a number
-- | signals an operation
-- @ signals a function
-- = signals an assignment

RPN = RPN_Expression
Ptr = 2 -- position after first special symbol.
-- SH is the stack height, EOL is End Of Line
INTIALIZE: SH = 0
LOOP <Evaluate>
BEGIN
     CASE RPN[Ptr - 1] OF -- special symbol
          WHEN ! => Get variable index
                    SH += 1
                    Stack[SH] = data from variable
          WHEN # => SH += 1
                    Stack[SH] = extracted number
          WHEN | => Operation = extracted operator symbol
                    CASE Operation OF
                         WHEN + => Stack[SH] += Stack[SH-1]
                                   Drop stack; SH -= 1
                         WHEN - => Stack[SH] -= Stack[SH-1]
                                   Drop stack; SH -= 1
                         WHEN * => Stack[SH] *= Stack[SH-1]
                                   Drop stack; SH -= 1
                         WHEN / => Stack[SH] /= Stack[SH-1]
                                   Drop stack; SH -= 1
                         WHEN CHS => Stak[SH] = - Stack[SH]
                    END CASE
          WHEN @ => Extract function name (or number)
                    Stack[SH] = function(Stack[SH])
          WHEN = => Get variable index
                    Store Stack[1] in variable index
                    Ptr = EOL
     END CASE
     IF Ptr = EOL THEN EXIT <Evaluate>  END IF
     Ptr = location of next space + 2
END LOOP <Evaluate>
TERMINATE: None
-- Result is in Stack[1] and variable appearing on the LHS of
-- the equation
END Interpreter
END PPL
```

Listing 2.

Add EDITING to your Software with CSE Run-Time™

Your program can include all or a portion of the *C Screen Editor* (CSE).

CSE includes all of the basics of full screen editing plus source in C for only $75. For only $100 more get CSE Run-Time to cover the first 50 copies that you distribute.

Use capabilities like Full cursor control, block move, insert, search/replace or others. Portability is high for OSes, terminals, and source code.

Call for the "CSE Technical Description" and for licensing terms and restrictions.

Full Refund if not satisfied in first 30 days.
Call 800-821-2492

Solution Systems™
335-L Washington Street
Norwell, MA 02061
617-659-1571

CIRCLE 93 ON READER SERVICE CARD

PROFESSIONAL PROGRAMMER'S BULLETIN:

Be Productive, Be BRIEF™

The Programmer's Editor

TRY BRIEF "RISK-FREE" FOR 30 DAYS WITH OUR MONEY-BACK GUARANTEE!

BRIEF's power and flexibility provide dramatic increases in programming productivity. BRIEF's ergonomically designed human interface becomes a natural extension of your mind, allowing you to eliminate tedium and concentrate on creativity.

- **WINDOWS**
- **Full UNDO (N Times)**
- **Compile within BRIEF**
- **Keystroke Macros**
- **Exit to DOS inside BRIEF**
- **Programmable Macro Language**
- **Multiple files, unlimited size**
- **"Regular Expression" search**
- **Reconfigure keyboard**
- **Language sensitive user controllable features (such as Auto-Indent for C)**

AVAILABLE FOR PC-DOS, IBM-AT, AND COMPATIBLE SYSTEMS

ONLY $195.

DEMO AVAILABLE FOR ONLY $10
(applicable to future purchase)

CALL TOLL FREE
800-821-2492
for "Technical Description" or to order.

Solution Systems™
335-L Washington St., Norwell, MA 02061
617-659-1571

BRIEF is a trademark of UnderWare.
Solution Systems is a trademark of Solution Systems

CIRCLE 97 ON READER SERVICE CARD

C Helper™
FIRST-AID FOR C PROGRAMS
Save time and frustration when analyzing and manipulating C programs. Use C HELPER's UNIX-like utilities which include:

DIFF and **CMP** – for "intelligent" file comparisons.
XREF – cross references variables by function and line.
C Flow Chart – shows what functions call each other.
C Beautifier – make source more regular and readable.
GREP – search for sophisticated patterns in text.

There are several other utilities that help with converting from one C compiler to another and with printing programs.

C Helper is written in portable C and includes both full source code and executable files for $135 for MS-DOS, CPM-80 or CPM-86. Use VISA, Master Card or COD.

Call: **800-821-2492**

Solution Systems™
335-L Washington Street
Norwell, MA 02061
617-659-1571

CIRCLE 99 ON READER SERVICE CARD

PROLOG-86™

Become Familiar in One Evening

Thorough tutorials are designed to help learn the PROLOG language quickly. The interactive PROLOG-86 Interpreter gives immediate feedback. In a few hours you will begin to feel comfortable with it. In a few days you are likely to know enough to modify some of the more sophisticated sample programs.

Sample Programs are Included like:

- ■ **an EXPERT SYSTEM**
- ■ **a NATURAL LANGUAGE INTERFACE** (it generates a dBASE II "DISPLAY" command)
- ■ **a GAME** (it takes less than 1 page of PROLOG-86)

PROTOTYPE Ideas and Applications QUICKLY

1 or 2 pages of PROLOG is often equivalent to 10 or 15 pages in "C" or PASCAL. It is a different way of thinking.

Describe the FACTS and RULES without concern for what the computer will have to do. Maybe you will rewrite in another programming language when you are done.

Programming Experience is not required but a logical mind is. PROLOG-86 supports the de facto STANDARD established in "Programming in Prolog."

CONTEST: Win $1,000. Ask about it. Deadline of 4/30/85.

AVAILABILITY: PROLOG-86 runs on MSDOS, PCDOS or CPM-86 machines. We provide most formats. The price of PROLOG-86 is **only $125.**

Full Refund if not satisfied during first 30 days.
800-821-2492

Solution Systems™
335-L Washington Street
Norwell, MA 02061
617-659-1571

CIRCLE 38 ON READER SERVICE CARD

WHITESMITHS, LTD. 370 COMPILER
NOW YOU CAN RUN BOTH C AND PASCAL ON IBM MAINFRAMES

FEATURES:

- Full implementation of the C programming language for the IBM 370/43XX/30XX
- Supports full ISO Level 0 Pascal, extended to support separate compilation
- Uses standard IBM tools to facilitate debugging in the standard IBM save area layout
- Compatible with the full range of Whitesmiths' compilers (PDP-11, VAX, 68000, 8080, and 8086)
- Provides optional cross support for MS/PC-DOS, CP/M-86, CP/M68k, and CP/M-80
- Runs under all versions of OS (VM, MVS, SVS, MVT, and MFT)
- Includes unlimited use of libraries in binary form
- Runs interactively under TSO and CMS
- Includes first year of technical support

CALL WHITESMITHS NOW AT (800) 225-1030

MASS. RESIDENTS CALL (617) 369-8499

WHITESMITHS, LTD.
97 LOWELL ROAD
CONCORD, MA 01742
TELEX 750246

Distributors: **Australia,** Fawnray Prance Ltd., Rosebery 2018, N.S.W., Sidney 662-4111
Japan, Advanced Data Controls Corp., Chiyoda-ku, Tokyo (03) 263-0383
United Kingdom, Real Time Systems, P.O.B. 70, Douglas Isle of Man, 011-44-0642-26021
Sweden, Unisoft AB, Goteborg, 011-46-31-125810

CIRCLE 43 ON READER SERVICE CARD

DESIGNERS DEBATE

Hashing out Forth with Charles Moore

By Ken Takara

This month, we present a discussion of various views on Forth. We examine some points for which it has been criticized, various topics internal to the Forth community, the applicability of Forth for different types of applications, and the peculiar nature of Forth as a programming language.

Among the participants we have Charles Moore, originator of Forth; Don Colburn and David Butler of Creative Solutions Inc. (CSI), a Forth vendor; Harry Wilker, vice president of development at Sentient Software, a software developer in Aspen, Colo.; Gregg Parsons, a programmer; and Dennis Hamilton.

Let's look at the technical aspects of Forth—in particular, the reverse Polish notation (RPN) and the parameter stack. Many people feel that the postfix notation makes Forth particularly difficult to read. You put something on the stack, and then some time later you use it. By then you've forgotten where it came from. This can make maintenance very difficult.

Moore: That's less of a criticism than it used to be. It has been addressed often enough. It's really the fault of your high-school teacher.

Infix notation is very puzzling to people. You learn it in high-school algebra. It's difficult, and once you finally master it, you want to feel it was worth the effort. You are unwilling to believe that it was a terrible mistake that you were forced to learn this notation. I think it was a terrible mistake— certainly for computer programming, maybe even for mathematics. Maybe one day people will quit having to learn this thing in high school and will get rid of this mind-set that makes them impatient with postfix notation.

The problem is amusing because Hewlett Packard has been selling RPN calculators for 20 years, and they've encountered exactly the same problem with it. Everyone else uses the "=" sign and infix notation, and they've captured the market. Hewlett Packard, for all that they were right, did not get the point across. I'm afraid Forth is in the same situation.

There's nothing wrong with postfix notation, and it's demonstrably superior to infix. However, it is unfamiliar, and so people will resist it.

It's not the case that Forth took a shortcut and just decided to cop out and use postfix notation. It's just that postfix is superior. When you encounter the operator in infix, it makes the presumption that its other argument is going to follow.

It has to somehow be able to pick up that argument no matter how complex it might be. In postfix the arguments are on the stack no matter where they came from. This gives you very nice decoupling because very often they come from nowhere. Postfix is the means we use for information hiding.

For example, I'm working on a Forth chip in which the parameter stack is attached to 16 pins on the chip, presumably attached to external memory. It might be attached to a signal source. Your signals might come to the stack without having been put there.

This is the value of postfix notation. There's no presumption as to how the data got on the stack. Yet with infix, the operator has got to somehow look ahead and figure out what its second argument is. And that requires arbitrarily deep, complex subroutine calls, and it gets very, very messy.

It appears that Forth itself is not so much postfix as a stack-based language. Things like *DUP*, *SWAP*, *ROT*, and *DROP* are specific to stack manipulation.

Hamilton: The stack in C is part of the mechanism, and its housekeeping is automatic. You don't have to plan anything about stack usage in order to develop C programs or libraries.

In Forth, which is by definition a lower-level language, the stack mechanism is part of the linguistic burden. You have to explicitly account for its operation in order to know how to define a procedure and to properly use a predefined procedure.

Any stack mechanism in C, Pascal, LISP, and other ALGOL-related schemes is hidden underneath the language as simply the way things get done. For example, my TI-52 algebraic-notation calculator has a stack inside, but I rarely think about it.

In Forth you have to program at the machine-language level of a synthetic stack machine. The machine language is rather interesting but actually not very powerful.

Moore: Yes, other languages have stacks, but they're never discussed, and they're never explicitly under the control of the programmer. They're a magic tool for systems programmers or for language designers. That's not right. There shouldn't be magic tools that only the rich classes can use. Why can't ordinary programmers have access to the same tools?

It seems, though, that one of the criticisms is that Forth makes it too easy for the programmer to bring down the system, usually in totally obscure and sinister ways. To many people, having the system just do whatever you told it to do without question is an invitation to disaster.

It is very difficult to track down the reason for a failure when anything is permissible.

Wilker: Forth demands excellence of the programmer. In return for that excellence, it delivers great performance. It's a very high-powered tool, and it's easy to stub your toe.

It seems to me that a language like Pascal enables mediocre programmers to write programs. For that purpose, it's a good tool. I'd hate to see mediocre programmers using Forth for professional development.

Pascal is like a house-building tool. It's like on late night TV, where this guy with a moustache and silk suit comes on and says, "For $49.95, we'll give you this incredible house-building tool. You hit this button and turn this lever, and up comes a stair-building tool. You use it and get a flight of stairs. And here's your wall tool and your kitchen sink tool."

What's happened, of course, is that the individual who wrote it has made all of my decisions for me. He or she says, "You

NEW FEATURES
(Free update for our early customers!)
- Edit & Load multiple memory resident files.
- Complete 8087 assembler mnemonics.
- High level 8087 support.
  Full range transcendentals
  (tan, sin, cos, arctan,
  logs and exponentials)
  Data type conversion and
  I/O formatting.
- High level interrupt support.
  Execute Forth words from within machine code primitives.
- 80186 Assembler extensions for Tandy 2000, etc.
- Video/Graphics interface for Data General Desktop Model 10

HS/FORTH

- Fully Optimized & Tested for:
  IBM-PC IBM-XT IBM-JR
  COMPAQ EAGLE-PC-2
  TANDY 2000 CORONA
  LEADING EDGE
  (Identical version runs on almost all MSDOS compatibles!)
- Graphics & Text
  (including windowed scrolling)
- Music - foreground and background
  includes multi-tasking example
- Includes Forth-79 and Forth-83
- File and/or Screen interfaces
- Segment Management Support
- Full megabyte - programs or data
- Complete Assembler
  (interactive, easy to use & learn)
- Compare
  BYTE Sieve Benchmark jan 83
  HS/FORTH 47 sec BASIC 2000 sec
  w/AUTO-OPT 9 sec Assembler 5 sec
  other Forths (mostly 64k) 70-140 sec

FASTEST FORTH SYSTEM AVAILABLE.
TWICE AS FAST AS OTHER FULL MEGABYTE FORTHS!
(TEN TIMES FASTER WHEN USING AUTO-OPT!)
HS/FORTH, complete system only: $250.
Visa Mastercard
Add $10. shipping and handling

HARVARD SOFTWORKS
PO BOX 2579
SPRINGFIELD, OH 45501
(513) 390-2087

CIRCLE 47 ON READER SERVICE CARD

want this kind of stairway and this kind of wall and kitchen sink." Well I say, "Wait a minute; I want a spiral staircase." Now I've got to go in and defeat all of the decisions made for me.

Using Forth is like buying a good set of tools and saying, "OK, I know what I want to do." Then I do it. It's hard for someone who's interested in Forth to appreciate the other point of view, but I don't want to go around fighting Pascal's decisions on how the world looks. I think it's far more efficient to define how the world looks myself.

CSI: Forth provides more power to the programmer than most languages. With this power comes added responsibility. The difference is like the difference between an automatic-transmission automobile and a manual-transmission automobile. You have to change the gears on the manual transmission, but there is a dramatic performance gain.

The way that Forth handles the disk seems to be a permanent point of contention. The use of blocks for source code is seen by a lot of people as a hindrance. And of course, the unavailability of files makes it difficult to pass data between Forth programs and other utilities.

CSI: Forth is a language that was developed at a time when standard operating systems were not readily available. Thus, Forth does have the rudiments of an operating system so that it may be used stand-alone on a computer. More recently, Forth systems are implemented using the computer's host operating system for standard input/output functions.

Wilker: All our files are based on blocks. That could be seen as a limitation by the outside world. We didn't see it that way. We managed to make the program easy enough to use and sufficiently complete that so far we haven't received much flak about it.

Even when using DOS files, to conserve space you tend to do tricks. You make heavy use of pointers, packing, and various other techniques that make the files incomprehensible to the user anyway. So it seems to me the issue is providing exportable file formats.

Moore: The thing people perceive as the power of files is the ability to take the data, process it a bit, pass it to someone else, process it some more, and so on. To my mind that is a clumsy way to do what I can do without having to create all those intermediary files.

So the issue is not what is the nature of data or how to organize it but what you must do to be compatible with the larger world. And what is the end value of it? I think the value is very low, and the cost is very high. If I had a little program that did something to my data—for example, one that sorts it—I would write a sort package and include it in my Forth system and forget about compatibility. But that's my personal attitude.

To become generally accepted, Forth is going to have to make contact with the greater world out there of millions of computers and millions of users. There are several ways of doing this, once you accept the need for compatibility.

One way is to write a utility that will take a Forth file and convert it to a DOS file and back again. This is fairly awkward if you're going to do it a lot. Another way is to take Forth files (sequences of blocks) and organize them to be compatible with the operating system. The problem with that is it's expensive. It costs in performance.

Increasingly, it is difficult to find a naked system where there isn't already an investment in software which you can take advantage of. The classic case is the Forth chip where there is no software except a prototype Forth system. If someone wants to add files on top of it, they will have to pay the price of the added complexity.

I'm sure they'll do it, but at least they'll be aware of the cost. Philosophically, I'm more concerned that people are aware of the cost. I don't want them to think it's easy simply because someone else did all of the work. It's a question of the true social cost being accounted for. It's a good issue.

Then, there is the matter of the source code being in blocks or files . . .

Moore: It is not an oversight that the model for the system is blocks. Blocks are superior to files in almost all regards. But that has to be carefully taught. It's got to be debated and discussed, and you've got to convince people of that. The natural reaction is that source programs come in files of arbitrary length. They keep returning fondly to that dream, saying, "Gee, if we only had files of Forth source, how nice it would be."

I remember one of the papers presented at FORML on this subject. It really offended me, though not so much the subject as the person who presented it. I've seen this same reaction in a number of other people who were discussing this subject. It's not that they say, "I just put my source in files and I find it more convenient." They say, "I just put my source in files and you should too. In fact, everyone should. This should be the new standard."

If you are on a PC under an operating system, and if you like to use its standard editor, then there are a lot of reasons to go with source in files. There are a lot of convenience factors. You can pick them out, archive them, edit them. You can do all kinds of things.

But there are a lot of things you can't do. For instance, given that you're in one file, can you load something which is contained in another file? Can files refer to each other in a nested fashion? Usually, the answer is no.

The problem of putting source in files is that it violates the modularity, which is a strong point of Forth. If you write a few words in this application, you can't reach out and pick them out for a different application because they're buried in the middle of a file.

You have to go in and reedit things and rearrange them and copy things from here to there. You end up with a real text-processing problem. And you usually end up with many copies of the same thing. Then if you change one, you have to change all of them, and they get out of synch. Then you have a library-maintenance problem.

With most languages you end up with a lot of straight-line code, even though there are subroutines. They are relatively few, and they're all stuck at the top of the program. I think the file structure encourages such straight-line code. If you can write a definition that is longer than one block, you probably will. Blocks make it difficult, or at least unattractive, to write long definitions. So they aren't written. And that is the modularity that Forth is trying to encourage.

CSI: As far as source screens go, even with an adequate file structure it is necessary to provide some mechanism of incremental compiling of Forth words. For most experienced Forth programmers, the Forth block is a convenient-sized chunk of code with which to work. Thus, even Forth implementations that use files may still preserve the Forth blocks organization of those files as an incremental compiling convenience.

Libraries are a function of the operating system. UNIX is very strong in this regard, while CP/M-68K is weak. Apple Macintosh has folders, a kind of poor man's library mechanism. Forth systems using files will automatically be able to take full advantage of whatever library features are available.

Moore: I approve of sequences of contiguous blocks. In fact, most file structures have blocks underlying them. I just question whether the notion of a file as an entity is worth what it costs. I keep trying to make people aware of the cost of what they're using. The fact that it's buried in the operating system doesn't mean you can ignore it.

I use the word *THRU* a great deal. I'll go out of my way a bit to move blocks around so I can use *THRU* instead of branching off to another block. But I still value the discipline of putting related words in the same block and the potential ability to pick up that block and use it out of context.

How is Forth for doing large, multi-programmer projects?

CSI: Forth is excellent for large, team applications because it has such a simple, elegant, parameter-passing mechanism. The project leader can divide the project into modules and define the parameters to be passed between them. The individuals are free to concentrate on their part of the problem. It is more a question of project management, design strategies, and documentation standards than of the particular language.

Moore: That is not exactly a wide open question, though the answer is perfectly clear in my mind. Yes, Forth is superb for such an application, but no, there should not be large programs. As soon as you realize that you are writing a large program, you should start asking yourself serious questions as to why it is so large.

There are business systems I've seen with huge source files of 2,500 blocks of source. It's a problem maintaining 2,500 blocks of code. It's a serious problem. My answer is not that you should develop appropriate management techniques to do it. You should get rid of the 2,500 blocks and reduce it to manageable size.

It's harder to write small programs. You have to put more thought into it. However, I think one of the keys to successful management of large projects is to make them into smaller projects. Either factor them into individual parts, or realize that it isn't as hard as you're afraid that it might be.

You gain maintainability in the sense that if someone else comes in and wants to figure out what you're doing, it's a lot easier to do when it is small. Even though you can afford to write a big program, and there's enough memory to hold it, that's the slightest part of the cost.

The industry trend is toward larger programs. It's as if people are paid according to the size of the programs they write rather than the value of the programs. It's natural enough. What measure of diffi-

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

WHICH FORTH has all the POWERFUL APPLICATIONS?

- **DATAHANDLER** database
- **FORTHWRITE** word processor
- **FORTHCOM** communications
- **GENERAL LEDGER** accounting
- **GAMES** for fun and technique
- **EXPERT-2** expert system
- **TRADESHOW** commodities terminal
- **GRAPHICS, 8087** support, many other utilities

You've Been Thinking About It. Isn't It Time to Put It to Work?

MMSFORTH

The total software environment for IBM PC, TRS-80 Model 1, 3, 4 and close friends.

- Personal License (required):
  - MMSFORTH System Disk (IBM PC) $249.95
  - MMSFORTH System Disk (TRS-80 1, 3 or 4) 129.95
- Personal License (optional modules):
  - FORTHCOM communications module .... $ 39.95
  - UTILITIES 39.95
  - GAMES 39.95
  - EXPERT-2 expert system 69.95
  - DATAHANDLER 59.95
  - DATAHANDLER-PLUS (PC only, 128K req.) 99.95
  - FORTHWRITE word processor 175.00
- Corporate Site License Extensions .... from $1,000
- Some recommended Forth books:
  - UNDERSTANDING FORTH (overview) ... $ 2.95
  - STARTING FORTH (programming) 18.95
  - THINKING FORTH (technique) 15.95
  - BEGINNING FORTH (re MMSFORTH) ... 16.95

Shipping/handling & tax extra. No returns on software.
Ask your dealer to show you the world of MMSFORTH, or request our free brochure.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

CIRCLE 49 ON READER SERVICE CARD

culty do you have other than size? If I have a big program, and you have a little program, I'm going to get more credit than you.

When you have people who spent a hundred man-years writing a program, and you tell them they didn't need to do it, they aren't going to be receptive.

It seems that, for some reason, some people are totally turned off by Forth. It's as though it has some sort of bad reputation. For example, John Dvorak, in his Inside Track in the October 29, 1984 issue of *InfoWorld*, made some deprecating remarks about Forth. He didn't back them up too well, so there must be others who take a similar view.

Wilker: I think there are two factors involved here. One is inherent in the nature of Forth itself. The other is the sociology of Forth as a community.

To be appreciated, Forth requires a paradigm shift. It requires you to see software development and programming from a new point of view.

FORTRAN, Pascal, and even C all share a fundamentally similar approach to programming. Forth says, OK, I want you to forget all of that. Forget algebraic notation. Forget procedures. Forget compiler security. Forget all that stuff and do it my way. That's both threatening and somewhat painful to people who think, quite rightly, that they know what they're doing. Added to this, people are hostile to Forth because they see it as a cult.

Parsons: I am new to the world of Forth, and I have only recently met programmers through some local user group meetings. I find the Forth people generally friendly and helpful, but there is definitely something very weird about this bunch that I can't put a finger on. I am very applications oriented; I think they seem to be into the language for itself.

These people are friendly and communicative up to a point, but then they suddenly turn cold and hard. Is this a reflection of the language, perhaps? I hope not.

Yes, I do know other programming types, so I have some basis for comparison. One annoying problem is trying to establish and maintain communication. In the middle of a conversation you find that you are not talking about the same thing. If you have a conversation going, they suddenly vector themselves out to places unknown.

Moore: Forth partly created its own problem by being so different. If it could have been less different, it would not have raised so many flags in peoples' minds. But in each case, the difference is justifiable as objectively advantageous.

Forth is a filter. Because it is so different, it has appealed to the fringe, the wild-eyed genius. It is going to appeal to people who are looking to be different. These people are going to be different in lifestyle as well as programming style. They are going to have sandals and beards, and this is going to offend the establishment. That's part of the motivation. So yes, there is an image problem.

Wilker: The reaction to Dvorak's article is interesting. I was down at Asilomar, and everyone was fuming about it. But very few people did what we did. We sent him a copy of our product. We said, "Here's a product that we think is good, and it's written in Forth."

Think of what would happen if 300 or 400 good Forth programs descended upon him! As for the 150 vitriolic letters he received, the results speak for themselves.

Unfortunately, Forth programmers are so in love with their tools that they don't turn out enough applications, and nobody hears about those they do turn out. They control spaceships or telescopes. They automate entire assembly lines and do scientific experiments.

WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing and carefully protects your text. Includes many features missing from WordStar, such as sorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily changed. Detailed manual included. Dealer inquiries invited. WRITE is $239.00.

BDS's C Compiler

This is the compiler you need for learning the C language and for writing utilities and programs of all sizes and complexities. We offer version 1.5a, which comes with a symbolic debugger and example programs. Our price is (postpaid) $130.00.

Tandon Spare Parts Kits

One door latch included, only $32.50.
With two door latches $37.50.
Door latches sold separately for $7.00.

All US orders are postpaid. We ship from stock on many formats, including: 8", Apple, Osborne, KayPro, Otrona, Epson, Morrow, Lobo, Zenith, Xerox. Please request our new catalog. We welcome COD orders.

Workman & Associates
112 Marion Avenue
Pasadena, CA 91106
(818) 796-4401

WORKMAN & ASSOCIATES USER-FRIENDLY SOFTWARE

CIRCLE 68 ON READER SERVICE CARD

The Tools You Need To C You Thru.

Now the WizardWare™ Applications Programmers Toolkit provides everything you need to increase your C programming productivity.

APT™ features include:

- COMPLETE SOURCE CODE (over 5000 lines!)
- File handling with direct & keyed access
- Screen and Report Generators, with full screen handling for your programs
- Generic Terminal Driver for portable code
- String math functions, and string manipulation routines
- Reference Manual on Disk (over 50 pages)
- Tutorial Manual (over 25 pages) with Source for Mailing List Manager
- A host of useful Utilities, Database and File Editors
- Available for Lattice C, Mark Williams C, DeSmet C, BDS C, others.

Also Available: C-STARTER Toolkit, great for learning C!! Includes: Customized APT, DeSmet C Compiler, and "Programming in C on the IBM-PC" (200 pages)

APT/MS-DOS versions	$495
APT/DeSmet C version	$395
APT/BDS C version	$395
C-Starter (binary APT, DeSmet Compiler and Book)	$295
APT/Manual only	$ 50

Detailed Brochures on request

*Manual Cost will be applied if APT purchased within 30 days ($10 re-stocking charge.) U.S. funds only, please.

Trademarks: MS-DOS/Microsoft, Lattice C/Lattice, Inc., MWC86/Mark Williams Co., DeSmet C/C Ware, CI-C86/Computer Innovations, Inc., BDS C/BD Software, DR-C/Digital Research, WizardWare, APT, C-Starter/Shaw☆American Technologies.

Call (502) 583-5527
Ask for APT™ or C-Starter, or Send Check to:
Shaw☆American Technologies
WizardWare™
830 South Second St. - Box 648
Louisville, KY 40201, USA
(C.O.D. and Foreign Orders - Add $5 Shipping/Handling),
References: Bank of Louisville, Citizens Fidelity Bank, Louisville Chamber of Commerce

VISA master charge

CIRCLE 34 ON READER SERVICE CARD

But what I regard as the simpler applications, the commercial and retail applications—there haven't been enough of them. People don't want to sit down and do them.

CSI: Due to its interactivity, extensibility, and flexibility, Forth lends itself handily to research topics in computer science and elsewhere. Likewise, Forth may be used to offer a general tool that is most intellectually stimulating to the programmer.

However, in industry, programmers are typically paid for results, not for tools. That is not to say that writing tools is necessarily bad. A well written tool will increase productivity, not decrease it. It is a matter of balance. If Forth has a fault in this matter, it is only that it amplifies both the programmer's intent and his or her ability to follow through with that intent.

Moore: There's another effect. Forth appeals to people who want results and don't care how they get them. If they can write a program in Forth and get it working, installed and paid for more quickly than they can in Assembler or BASIC, they'll use Forth.

Those result-oriented people tend to cut corners. They may not fulfill every detail in the contract and expect to get paid. So there's a quality problem—the attitude: it works well enough, so I'm done. They get impatient with the tedious, exponential tail of a project, which, to get exactly right, takes ten times longer than they anticipated.

Forth is not the product of the respected, meticulous software industry but is sitting somewhere on the fringes where the reputation, credibility, and performance of the people using it might not be what you've hoped.

Wilker: Another reason is that nobody is making money on it. There's the UCSD Pascal, and Turbo Pascal. There are people making money with Pascal. Few people make money selling Forth systems. Forth is so easy to implement that everyone sells systems for sixty bucks, then provides unlimited phone support as a kind of religious experience.

Forth would probably get more respect if there were more real companies charging a more realistic price for their product and then supporting it in a professional manner.

CSI: Many early Forth vendors' products were based on public domain source code and were sold at prices that were unable to sustain adequate documentation and product support. Today, this is not the case. Most successful Forth vendors offer full support of their products.

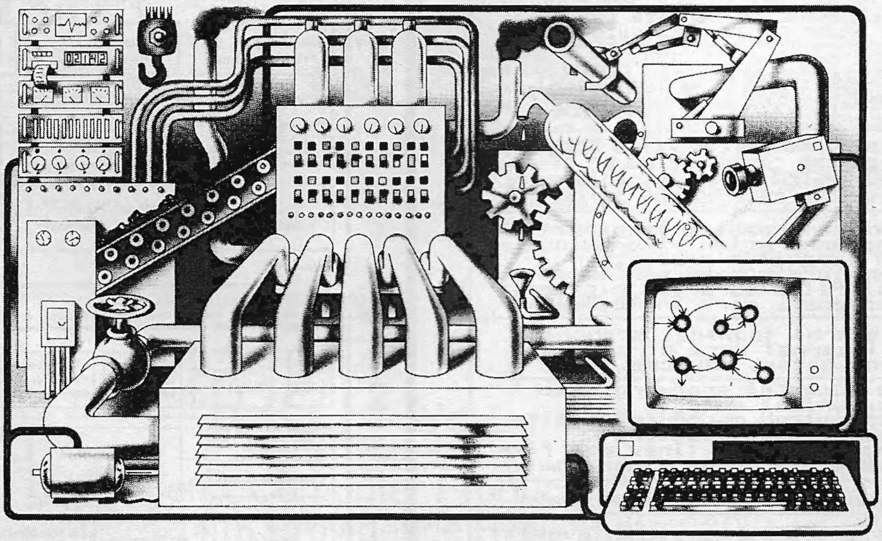
Wilker: We also don't have an academic constituency. Academics love Pascal. That gives it respectability. It doesn't bother them that those of us who are out in the real world producing applications can't use it. On the other hand, Forth was developed by practitioners needing to accomplish specific objectives.

Moore: Forth has made its greatest progress in the engineering departments of universities. It is used in astronomy, physics, and chemistry. It is rarely used in the computer science department.

The support we do have is suspect. It's not the official university computer center. It's the people with the PCs trying to get their equipment to work. It is offensive to the establishment for people to go against their wishes. They say, why can't they use Pascal in the engineering department? It is a perfectly good language; we're using it all through the university, except in this one bloody department.

That department is quite happy with Forth, but that is not helping Forth. It is probably hurting it because everyone else is all of a sudden focused on it, saying that's what they don't want to have happen. I'm at a loss as to why that's so, but it certainly is. Forth has lacked any kind of

Hard problems? **Csharp** can help!



Cut your development time with C source code for realtime data acquisition and control. The **Csharp Realtime Toolkit** includes: graphics, event handling, scheduling, and state systems. Processor, device, and operating system independent. Price: $600

SYSTEMS GUILD

Systems Guild, Inc., P.O. Box 1085, Cambridge, MA 02142
(617) 451-8479

CIRCLE 89 ON READER SERVICE CARD

substantial university, corporate, or institutional support. Yet it's amazingly successful without that support.

CSI: There have been some problems with standardization. Nevertheless, interest in and use of Forth is growing exponentially each year. Standards are being developed, and vendor support has improved dramatically.

Wilker: Portability was a major factor in our decision to use Forth. We have to support our Back to Basics Accounting product on the IBM PC, the PCjr, other MS-DOS machines, the Apple II line, the Atari, the Commodore 64, and now on the Macintosh.

Moore: We've done very well in the hobbyist market, but that shouldn't be the major thrust. We need to go to the government and commercial market. That's why standards exist. We want the respectability. We want to be recognized as a viable programming language so we can be involved in nice, big projects. A lot of nice projects out there can't be written in Forth because they look at you funny and say, "Well, we do it in Ada."

I don't want to be involved in any kind of standardization effort. It leads to respectability, but I'm on the other side of the wall. I want to take the standard as a starting point and branch out from it. Try all these different, nonstandard things and see if they're more useful.

I'm really a very nonstandard person, except that I appreciate the standards in the sense that if we didn't have a standard, then no one would give us the time of day.

It's useful for portability and respectability, but there's nothing wrong with being nonstandard as long as you don't go public with it.

Wilker: It's an interesting testament to Forth that much of the significant software for the Macintosh will be in Forth.

In a new environment like the Mac, the ability to poke around interactively is an incredible advantage. We'd still be trying to figure out how to use the damned machine if we were using Pascal. The process of development would be extremely tedious given the necessity to compile, link, test, recompile, link, etc. Instead, in a matter of hours, we were getting results. We could poke a value here, watch the machine blow up and say, "Oops, not there," and go back and try it again.

Forth's interactive ability is recognized as its major strength, and it's peculiar syntax is considered it's greatest weakness. Perhaps one of the big problems is the general unfamiliarity with the Forth language.

Moore: Familiarity is a fairly transient objection once you become familiar with it. I've seen this happen with many students of Forth. It may take them a week to get over the hurdle of how the stack works, but once they do, they can't even remember what the problem was.

FORTH Inc. put Leo Brody to work writing *Starting FORTH* very soon after he had learned Forth, before he had forgotten the problems he had learning it. Experienced Forth programmers tend to be unsympathetic to the problems the novice has. Those difficulties are a one-time problem. Once you overcome them, they don't have to recur.

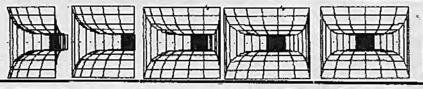
A programmer often finds a language in which he or she can express themselves more eloquently than in any other language. Somehow, the code is much more aesthetically pleasing, both to read and to write. The programmer sometimes finds that he or she can, in fact, think in that language.

Do you have a language that you particularly like for its expressive qualities, something beyond the usual claims of efficiency or mere maintainability? For example, a Forth programmer might be particularly fond of a sequence of actions for a robot expressed in Forth.

If you do, send in a letter describing why you find the language of your preference aesthetically pleasing to write in. Send in an example of a subroutine you have written that exemplifies what you consider elegant coding in that language.

Mail it to: Language Aesthetics, c/o *COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, Calif. 94107. ∎

What else could you do, if you had a 3-D GRAPHICS LANGUAGE to harness the computational power of your computer?



E.G. is a new programming language that provides most of the computing power of a dedicated CAD SYSTEM at an incredible price: less than one cent on the dollar

E.G. is a complete graphics language:
* automate the generation of drawings
* design math intensive products
* unwrap & template curved surfaces
* output ascii files to drive n.c.tools
* accuracy exceeds 1 part in 32,000

43 commands control looping, branching disc access, internal data structures, subroutines and output devices.

E.G. is also a powerful post-processor:
* plot your numerical data in 3-D
* virtually unlimited macro capability
* write "turn-key" applications
* 1000 data points, 1000 lines per file
* optional "best fit" plot algorithm

Easy to use, rather than easy to learn

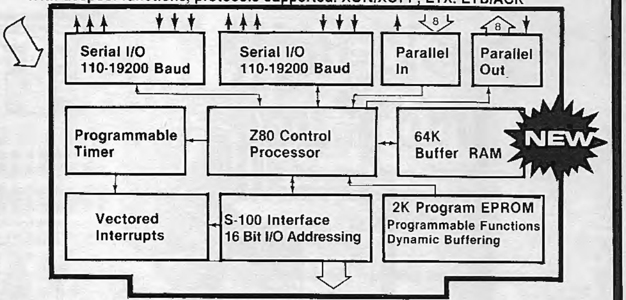
E.G. provides the programmer with every mathematical operation & function found in BASIC + powerful, proprietary shape generating and manipulating commands.

EUCLID GRAPHICS
SYSTEM: 64K APPLE D.O.S 3.3
developed at
SEVEN SEAS SOFTWARE
35 Cape George Wye
Port Towensend, Wa.
98368
(206) 385-3771
MANUAL $ 25
COMPLETE $200
free updates for one year

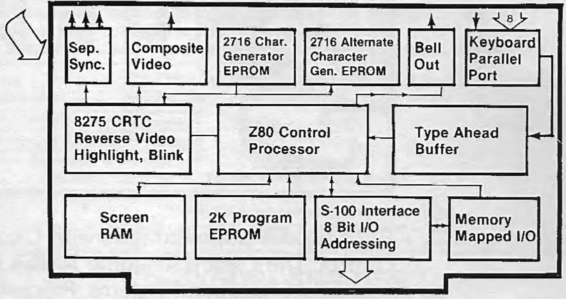
signed nondisclosure agreement required

CIRCLE 28 ON READER SERVICE CARD

BUFFERED I/O BOARD Introductory Price * $59.95
With despool functions, protocols supported: XON/XOFF, ETX: ETB/ACK


Serial I/O 110-19200 Baud
Serial I/O 110-19200 Baud
8
Parallel In
8
Parallel Out
Programmable Timer
Z80 Control Processor
64K Buffer RAM
NEW
Vectored Interrupts
S-100 Interface 16 Bit I/O Addressing
2K Program EPROM Programmable Functions Dynamic Buffering

80 CHARACTER VIDEO BOARD * $49.50
25 Lines with status, compatible with Wordstar & dBase


Sep. Sync.
Composite Video
2716 Char. Generator EPROM
2716 Alternate Character Gen. EPROM
Bell Out
8
Keyboard Parallel Port
8275 CRTC Reverse Video Highlight, Blink
Z80 Control Processor
Type Ahead Buffer
Screen RAM
2K Program EPROM
S-100 Interface 8 Bit I/O Addressing
Memory Mapped I/O

Includes Bareboard, Heatsink & Documentation Call or write for more information.

Simpliway Products Co.
P.O. Box 601
Hoffman Estates, IL 60195
(312) 359-7337

SPC

VISA

MasterCard

OEM dealer pricing available, $3.00 S/H, IL. Res. add 7% tax
dBase™ - of Ashton - Tate Corp. — Wordstar™ - of Micropro Int'l. Corp

CIRCLE 70 ON READER SERVICE CARD

POWER TOOLS

from Phoenix

You are a serious programmer. Your income depends on developing software solutions for complex problems. Fast. Error free. You need powerful tools for your compiler. That's what we sell.

PLINK ™ 86
OVERLAY LINKAGE EDITOR

- overlays for modules are automatically loaded during execution
- generates symbol tables for PFIX 86 PLUS
- accepts unlimited number of files
- capacity for over 35,000 symbol, commonblock and variable names
- change overlay structure without recompiling
- sorted memory map reports
- user defined width/height for reports
- define symbols as absolute addresses or offsets from other symbols
- **overlay is user configurable**

List: $395

PFIX ™ 86 PLUS
DYNAMIC SYMBOLIC DEBUGGER

- debugs PLINK 86 overlays
- view source code while debugging
- set temporary or permanent breakpoints
- advanced tracing and trapping capability
- in-line assembler for temporary patches
- preserves program screen and supports dual CRT configurations
- multiple windows to view code, data, breakpoints, register/stack contents, simultaneously
- supports 8087 numeric coprocessor

List: $395

PMATE ™
TEXT EDITOR PROCESSOR

- automatic disk buffering
- 10 auxiliary buffers that can be edited
- garbage stack
- user customization
- **120 command macro language**
- **store macros permanently**
- arithmetic and logical operations
- 100 numeric variables
- automatic formatting with indent and word wrap
- insert, overwrite and command modes
- horizontal/vertical scrolling

List: $225

[PC]^2 We Sell Power

Phoenix Computer Products Corporation
1416 Providence Hwy., Suite 220
Norwood, MA 02062
800-344-7200 (Mass.) 617-762-5030
Visa • Mastercard • American Express

CIRCLE 82 ON READER SERVICE CARD

Modifying MS-DOS Device Drivers

By Mike Higgins

In the past, only mid- to large-sized computers had operating systems that allowed loadable or user-written device drivers. MS-DOS is the first widely distributed personal operating system that brings the power of loadable drivers to home computers.

This feature of MS-DOS increases the range of devices available to users because OEMs now have a well-defined way to integrate their hardware into microcomputer systems. Anyone who is willing to write his or her own device drivers can add the devices in a transparent way, rewrite inadequate drivers, or write virtual device drivers for a specific system.

This article is aimed at both OEMs and everyday MS-DOS programmers. Anyone mildly proficient at 8088 assembly programming can probably write an MS-DOS device driver. The article will cover the reasons for writing device drivers and how they are integrated into the system. The different parts of a driver will be described, with special attention to some pitfalls that the MS-DOS device drivers documentation glosses over.

Finally, I will discuss techniques for debugging device drivers—techniques that work on ordinary MS-DOS systems without expensive test equipment. I'll describe a recent driver I wrote, a replacement driver for IBM PC COM ports that implements type-ahead, data throttling, and many other features that allow you to use the IBM COM port as a console terminal or hook it up to a wider variety of RS-232 devices.

MS-DOS vs. PC-DOS

What is the difference between MS-DOS and PC-DOS? How can you tell if a program is an MS-DOS program or a PC-DOS program? The basic difference is that a PC-DOS program depends on specific memory locations, port addresses, and ROM routines of a certain specific brand of 8088 system. If you want to write MS-DOS programs instead, just follow these simple rules:

1. Never under any circumstances use any 8088 INT instruction, except INT 21H
2. Never write to known memory locations
3. Never use the 8088 *IN* and *OUT* instructions
4. Never call any ROM routines. (Don't even read them!)
5. If you find yourself needing to break rules 1 to 4, then you should do it in a driver!

Situations will arise where you will decide to break these rules, but first consider the alternatives. If you can obey these rules, the software you write will have the largest possible audience. Many products on the market break these rules for no good reason—except maybe that "the ROMs were there"—and then fail to work when users upgrade to a hard disk or try to buy the software to run on another system. This can be avoided in many programs.

If you must break the rules, consider doing it in a driver first. Putting machine-dependent features in a driver allows you to isolate them from the up-front task of writing an application program. Just remember: inside every PC-DOS system is a powerful, flexible, MS-DOS system just waiting to be set free.

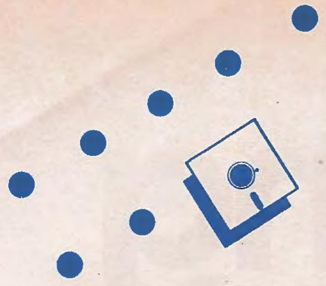
Why write a device driver?

The traditional reason for writing a device driver is to attach a new device to a system. If you are an OEM considering building a new device for the IBM PC market, you have perhaps three alternatives:

- Make your device look just like something that the ROMs already know how to read, which means you are competing with Big Blue
- Convince your users to break the rules (I would think long and hard before buying your product)
- Write a device driver. MS-DOS is extraordinarily well thought out, and it is possible to write drivers for a wide variety of devices, such as tape drives, memory banks, communications ports, network controllers, and many devices that don't really exist at all. (More on that later).

One of the really well-designed features of MS-DOS is the way a new BIOS for a totally new system is written—as a linked list of device drivers. Once you can write a device driver, you can write a new BIOS for an MS-DOS system and port it onto a totally new 8088 system. If you are an OEM building an MS-DOS system, you will need to learn how to write device drivers to build the BIOS.

I am very dissatisfied with the drivers IBM chose to supply with PC-DOS. The



ANSI.SYS driver, for example, is a console device driver that makes the IBM PC screen and keyboard look like "an ANSI terminal." Ha! It implements three or four of the ANSI terminal escape sequences and stops. The printer driver is another bone of contention. Because of the way it is written, the 1,000 cps parallel interface cannot keep up with an 80 cps printer.

The .COM driver in PC-DOS was so bad that I have rewritten it. The old .COM driver could not do XON/XOFF protocol correctly, could not do hardware handshaking right, reported error messages in inappropriate ways to MS-DOS, did not have a type-ahead buffer . . . and the list goes on.

Another reason for writing a driver is to produce virtual devices. The classical example is the RAM disk, but this idea is really old hat. The concept of a virtual driver can be applied to other ideas, for example, an 8087 driver. On systems that can get interrupts from their 8087 chips, you could write a driver that performs long calculations in the background.

Another driver extension idea would be a smart spooler driver that accepts large amounts of data and sends it to slower devices like printers. A graphics driver could be written that accepts NAPLPS codes and writes pictures on the bit mapped display. This idea could make it possible to write totally device-independent graphics programs. Just rewrite the NAPLPS driver for a new system and your program runs on that system.

What makes a driver virtual instead of real is that these drivers usually do not have a device attached to them. They use system memory or talk to other drivers instead. This means that a virtual driver has an even larger chance of running on any MS-DOS system than a real driver does.

Overview

Before writing a driver, it is helpful to know the steps involved in booting MS-DOS. When the 8088 system first boots, it executes a routine starting at location FFFF:0000 (this is a feature of the 8088 processor chip). Since hardware designers have no choice but to put the restart routine here, the rest of the system's ROM will probably be here as well. Usually the only code at FFFF:0000 is a far jump to a larger boot routine.

```
Recommended driver structure

 +-------------------------------------+
 |            DEVICE HEADER            |
 +-------------------------------------+
 |   DRIVER INTERNAL DATA STRUCTURES   |
 +-------------------------------------+
 |          STRATEGY ROUTINE           |
 +-------------------------------------+
 |          REQUEST ENTRYPOINT         |
 |(called the interrupt entrypoint by MS)|
 +-------------------------------------+
 |          FUNCTION ROUTINES          |
 +-------------------------------------+
 |      INTERRUPT SERVICE ROUTINES     |
 +-------------------------------------+
 |        INIT FUNCTION ROUTINE        |
 +-------------------------------------+
```

Figure 1.

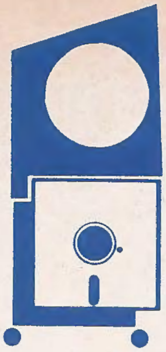
Microsoft recommends that the first block of a floppy disk be reserved for a boot program so this block may be read as the next step in booting a system.

Two files on a bootable disk are loaded next; these are the binary image of MS-DOS and the system-specific DOS driver, or BIOS. These two files are named MSDOS.SYS and IO.SYS, respectively, but since they are not read as files, they can be named just about anything. I believe IBM renamed them to IBMDOS.SYS and IBMIO.SYS, then marked them as invisible files so they never appear in directories.

It is the responsibility of the boot routine to know how to read these two files off of the disk, where to put them in memory, and how to start them. Starting them means jumping to an initialization routine in the DOS driver (BIOS). This initialization routine tells the MSDOS.SYS code how to find the rest of the BIOS (using some external variables provided) and jumps into MSDOS.SYS. If all goes well, MS-DOS calls all the individual driver *INIT* routines in the BIOS's list of drivers. At this point, MS-DOS can start using the BIOS to print startup messages and read data off the disk. (Before this, MS-DOS is totally deaf, dumb, and blind).

Once MS-DOS knows how to read a floppy, it opens several important files. The one of greatest interest to our discussion is the CONFIG.SYS file. Among many other things, the CONFIG.SYS file allows you to specify the file names of additional loadable drivers.

Usually the IO.SYS file (the BIOS) is read by a ROM routine that knows exactly where the file starts and how many blocks to read. It is not possible to add new drivers to this file because that would confuse the boot routines. This is why loadable device drivers are such a powerful feature—they are loaded after MS-DOS comes (part way) up and can be read as files by name. They can be any size, they can start anywhere on the floppy, and they can be fragmented like any other file. The commands in a CONFIG.SYS file are entered as simple ASCII text and can be



created with most text editors or even with the *COPY* command when talking to MS-DOS's command interpreter.

The following is a sample CONFIG.SYS file from my system:

```
device=qdrive.bin
DEVICE=ASYNC.BIN
```

The *DEVICE=* command tells MS-DOS the name of a loadable device driver. QDRIVE.BIN is a driver I purchased to handle an 8-in. floppy disk drive hooked up to my IBM PC. ASYNC.BIN is a driver I wrote to replace the COM1 and COM2 drivers that come with the PC.

You may wonder why the commands in the CONFIG.SYS file bear little resemblance to the normal commands typed to MS-DOS. MS-DOS's command interpreter is stored in another file, COMMAND.COM, and it is not in memory at the time that loadable drivers are read. The command interpreter is a separate program that is loaded after MS-DOS, and this opens up the possibility of writing new command interpreters for the system!

But that is the subject of another article. Just to mention it in passing, another file that is read while the system is booting is AUTOEXEC.BAT. This file is actually read by the command interpreter, so it does contain commands you normally type to MS-DOS.

Parts of a driver

A driver is a relocatable memory image similar to a .COM file. But instead of containing an executable program as a .COM file does, the driver file contains a special set of tables and routines that implement the device. These tables and routines must be in a specific order or MS-DOS will not be able to call the driver. Figure 1 shows how a driver is laid out. This diagram shows my preferred way to build a driver, but other combinations are possible.

Device header. Unlike a normal .COM file, (which is ORGed to 100H), a driver is ORGed to 0 and the device header must be the first thing in the file. This structure contains offsets to the other important parts of the file, so while you have some leeway where you can put those, this header absolutely must be at the start of the file. Listing 1 shows a fragment of MASM code that demonstrates the format of a device header.

The system uses this device header to find out who you are and where your routines are. The first two words are filled in by MS-DOS when it links the driver into the rest of the system. The device type word has bits for character device (as shown), block devices, standard input and output (CON device), NUL device, CLOCK device, and finally for *IOCTL*. *IOCTL* (Listing 2) is a powerful feature I will cover in more depth later.

The offsets to *strategy* and *request* are used by the system to find your routines when it wants to use the driver. The last thing in the header is an eight-byte device name string. You only need to name character devices—block devices (like disk drives) are assigned single letter names by the system. On character devices, the name can be any eight-byte legal file name. If you are implementing a new device, make sure this name does not conflict with any old ones. If you are writing a driver to replace an existing one, MS-DOS always searches the last character device name first, so loadable device drivers can supersede the ones the system comes with if you use a conflicting name on purpose.

In block devices, disks, etc., the first byte of the name field in the device header contains the number of units that the driver will be driving. The remaining seven bytes are ignored, so I like to put the driver name there. This is sort of a documentation trick that survives assembly, linking, conversion, and loading into memory. It helps you recognize the device driver when looking through memory with debug.

Driver internal data structures. When writing a driver, you need to allocate space for temporary variables, ring buffers, caches, baud rate conversion tables, or myriad other uses. I prefer to place this data immediately after the device header for several reasons. When you write a driver, you do not know where the code will be loaded into memory, so data and code segments will be combined together. The driver list head looks like data, so it seems reasonable to stick all data up there with it. Another logical place to put the data might be at the end of the driver.

But MS-DOS allows you to throw away initialization code off the end of the driver, so the *INIT* function routine takes that place. Of course data could be scattered in between all of the other routines, but this is not a recommended programming practice.

***Strategy* routine.** The *strategy* routine is an artifact from the future. MS-DOS currently does not handle multitasking, but the people at Microsoft are planning for it. In that glorious future, this *strategy* routine will be called to start a driver working on a *request*, which can be driven

```
;FRAGMENT OF A  DRIVER SHOWING AN EXAMPLE DEVICE HEADER

DRIVER        SEGMENT
        ASSUME        CS:DRIVER,DS:DRIVER
        DW        -1,-1               ;DRIVER LISTHEAD
        DW        8000H               ;THIS IS A CHARACTER DEVICE
        DW        STRATEGY            ;OFFSET TO STRATEGY ROUTINE
        DW        REQUEST             ;OFFSET TO REQUEST ROUTINE
        DB        "ASYNC   "          ;DRIVE DEVICE NAME
        . . .
```

Listing 1.

by interrupts in the background while other tasks run on the system.

For now, MS-DOS simply calls a *strategy* routine and then immediately calls a *request* routine (described later). According to the device drivers section of the MS-DOS manuals, the *strategy* routine is supposed to save the contents of the ES and BX registers for the *request* routine to look up later. I have seen people omit this step because they know MS-DOS does not change the contents of ES or BX before calling the *request* routine and this works. I assume it will fail in future versions of MS-DOS, so I always save these registers in the *strategy* routine and read them back in the *request* routine. Listing 3 is a fragment of a driver that shows a short generic *strategy* routine that can be used in any driver.

***Request* entrypoint.** In the MS-DOS driver documentation, there is a routine in drivers called the *interrupt entrypoint.* I object to this name strenuously, because an *interrupt* entrypoint is commonly understood to be the place that an interrupt vector points to, which is terminated by an *interrupt return* instruction. The routine that Microsoft calls the *interrupt entrypoint* of a driver is the entrypoint that MS-DOS calls (with a *far* call, not an *interrupt*) to make a *request* to the driver.

In the future multitasking versions of the system, this routine will serve a different purpose, but it is still misnamed. In all of my drivers, I have called this portion of the code the *request* routine. The system calls the *request* routine at a single point, but the static *request* header structure pointed to by ES:BX contains a command that describes which function the system wants the driver to perform. It is the responsibility of the driver to decode this command code and branch off to separate routines for handling each different code.

First let me share with you a way I got around a limitation in MS-DOS device drivers. As mentioned earlier, a block device (like a disk driver) is expected to control several units, but a character device driver is assumed to have only one. If you are writing a device driver that will

```
IOCTL.ASM

;PERFORM IOCTL CALL FOR C MAIN PROGRAMS
;(c) Copyright 1984 by The Computer Entomologist
;
;       This routine can be called from C programs to
;perform I/O control on any MS-DOS device driver that
;was written to accept it. This version here was written
;for the Computer Innovations C86 compiler.
;
;       Permission is hereby granted to use or distribute
;this software without any restrictions.  You may include
;it in any hardware or software product that you sell for
;profit.
;
;       If you have any questions about this software, you
;can reach me at the address below.  If you implement any
;new features or find (and fix!) any bugs, I would be happy
;to hear from you.
;
;       Mike Higgins
;       The Computer Entomologist
;       P.O. Box 197
;       Duncans Mills, CA 95430

PUBLIC  IOCTL   ;(ASYNC,BUF,LEN,FUNC)

@CODE   SEGMENT PUBLIC 'CODE'
        ASSUME  CS:@CODE

IOCTL   PROC    NEAR
;ARGUMENTS:
ASYNC=4 ;ADDRESS OF ASCIZ DEVICE NAME.
BUF=6   ;ADDRESS OF A BUFFER TO READ/WRITE FROM
LEN=8   ;THE NUMBER OF BYTES IN BUFFER
FUNC=10 ;2 FOR IOCTL READ, 3 FOR IOCTL WRITE

        PUSH    BP                      ;DO THE NORMAL C LINKAGE
        MOV     BP,SP

        MOV     DX,ASYNC[BP]  ;GET NAME OF DEVICE TO OPEN
        MOV     AX,03D02H     ;OPEN IT FOR READ/WRITE.
        INT     021H          ;WELL, CAN DOS DO IT?

        MOV     BX,AX         ;STORE HANDLE IN BX
        MOV     DX,BUF[BP]    ;GET BUFFER ADDRESS
        MOV     CX,LEN[BP]    ;AND LENGTH.
        MOV     AX,FUNC[BP]   ;GET FUNCTION REQUESTED,
        MOV     AH,68         ;TELL DOS THIS IS IOCTL CALL
        INT     021H          ;WAKE UP DOS.
```

Listing 2 *(Continued on following page).*

ultimately be used on several character devices, it is a shame to duplicate all that code in memory, and you would really like to have several units in the driver.

Also mentioned earlier was the fact that a BIOS was a linked list of device drivers in one file. You can still link several devices into one file in your own device driver, so I use this feature to link several character devices into one file that shares a lot of code.

In fact, if you make your driver code reentrant, you can share almost all the code. I accomplished this by placing all the data for each unit of my driver in a separate structure. Each unit has a separate *request* routine entrypoint and a separate interrupt service entrypoint that does only one thing: it loads a pointer to its unit structure into SI, then falls through to the normal driver code. Except for scratch variables, the normal driver code learns everything it needs to know about its device from this structure by referencing relative to [SI].

The driver documentation barely skims over the *IOCTL* capability of device drivers in MS-DOS. The ramifications of this powerful feature are not spelled out to my satisfaction. This function can be used to send to and from the driver information that does not go to the physical device.

The example given in the documentation is: use *IOCTL* to send baud rate, parity, stop bits, and frame length information to an RS-232 device driver. Apparently, that documentation was not read by the people who wrote the PC-DOS *MODE* command. That command seems to poke around in known locations, write to I/O ports, call ROM routines, and generally do a lot of nonportable things. If there had been a documented *IOCTL* write to the .COM ports on an IBM PC, then *MODE* could have been made into a portable MS-DOS program. In a cache or RAM disk, *IOCTL* can be used to tell a driver how much memory to use and where it is. (That information can be had from standard MS-DOS memory allocation calls).

I have gotten to the point where I want to have several *IOCTL* functions in the same driver, and I'm considering an additional jump table that uses the first byte of the *IOCTL* data as a sub-function code. Besides setting line parameters, I need to set special character handling bits, do a "connect to interrupt" *IOCTL* write that calls me back when the modem status lines change . . . All these things and more can be done within the framework of MS-DOS, and if they are documented, they can be written into a driver for different devices or 8088 systems without having to rewrite the large application programs that call them. *IOCTL* is a powerful tool—use it!

Interrupt service routines. I'm amazed by the number of devices on my system that are not interrupt driven. The MS-DOS device driver documentation mentions that the "DOS assumes all character devices have an input type ahead buffer," and I only wish that the .COM port drivers on the IBM PC actually did.

The parallel interface driver is not interrupt driven for output. If it were, it could have an output ring buffer that would unstop the bottleneck and allow it to send data as fast as my 80 cps printer could accept it. I have heard from some people at Davong Inc. that this omission is partly due to a hardware glitch in the IBM parallel adaptor—it does not always reliably fire an interrupt when the ACKNLG line goes low. The IBM PC has documentation problems with interrupts in general. The PC has only eight hardware interrupts, and an undocumented 8259A chip allows you to disable each one of these eight interrupt lines. If you don't know about this chip, you never get interrupts from your device because you didn't clear the appropriate bit in the 8259A's port at I/O address 20H and 21H.

The documentation for the IBM async card also fails to mention that interrupts from the UART are "ANDed" with a line called the OUT2 line. If this undocumented bit is not set, interrupts never make it to the undocumented 8259A. It seemed to me, while I was trying to write my driver, that the list went on forever—first you set the interrupt enable bits in the UART, then you set the OUT2 bit, then you clear a bit in the 8259A, and

```
          MOV     AH,62          ;SET UP TO CLOSE HANDLE WHEN DONE
          INT     021H           ;ASK DOS REAL NICE.

          POP     BP             ;RECOVER BP
          RET                    ;RETURN TO C

IOCTL     ENDP

@CODE     ENDS

          END
```

Listing 2 *(Continued from preceding page).*

```
;EXAMPLE GENERIC STRATEGY ROUTINE
            . . .
            HEADER:  DD     0        ;STORAGE FOR ES BX

STRATEGY          PROC    FAR
                  MOV     WORD PTR CS:HEADER,BX    ;SAVE OFFSET
                  MOV     WORD PTR CS:HEADER+2,ES  ;AND SEGMENT
                  RET
STRATEGY          ENDP
            . . .
```

Listing 3.

then you make sure to enable interrupts in the 8088 flags register. Maybe I know now why so few device drivers use interrupts on my system.

When I write a character device driver that shares code with several units, each unit must have its own interrupt service entrypoint. But again, the entrypoints can load the address of a structure and execute a common interrupt service routine. Listing 4 is a fragment of a driver that shows a common interrupt service routine being used for two character devices.

***INIT* routine.** The *INIT* routine is the first routine that MS-DOS calls in the driver. It is usually the first routine I write, but it is always the last routine in a driver. This is because MS-DOS calls this routine only once, and it allows you to throw the code away after that call.

When an *INIT* routine is called, the device driver documentation says you must return a thing called a "break address" in the request block. I had to call MS-DOS the first time to find out that break address means the end of your driver. If the break address is put before the *INIT* routine, the DOS will write something else there later and re-use the memory beyond that point.

I have successfully written character device drivers that had the same *INIT* routine called twice because two units shared the same *INIT* code, and MS-DOS did not get confused. Since finding that out, I have come across a version of the documentation that actually said this was OK, although it is recommended that all the device drivers in a single file return the same break address.

The same manual that mentioned multiple drivers also mentioned an extremely useful feature that I was totally unaware of. When MS-DOS loads a driver at CONFIG.SYS time, it loads a pointer in the break address of the request block. This points to the character just past the = on the line from the CONFIG.SYS file that loaded the driver. I haven't tried this, but it opens the possibility of another channel of communication to the driver.

When something is needed to be sent to the driver just once, it can be included as an argument in the *DEVICE=* command in the CONFIG.SYS file. The *INIT* routine could scan this line and get the information it needs from those arguments. An example would be a RAM disk driver that dynamically allocates its memory at *INIT* time. You can tell the driver how much memory to grab by including your own /SIZE:xxxx argument in the CONFIG.SYS command. This would eliminate the necessity of rebuilding a RAM disk every time the memory size needs to be changed (a failing I have seen in some RAM disk drivers). The extended memory driver written by IBM for the PCjr actually did use this feature to tell the system how much memory to reserve for the graphics display.

```
;FRAGMENT OF CODE SHOWING TWO INTERRUPT SERVICE
;ROUTINES SHARING THE SAME DIVER FILE

    . . .
ASY1ISR:
        PUSH    SI                  ;SAVE SI SO YOU CAN
        LEA     SI,ASY_TAB1         ;POINT AT UNIT 1'S STRUCTURE
        JMP     INT_SERVE           ;JUMP TO GENERIC INT SERVER
ASY2ISR:
        PUSH    SI                  ;DO THE SAME THING FOR
        LEA     SI,ASY_TAB2         ;UNIT 2.
INT_SERVE:
        PUSH    AX                  ;SAVE ALL THE REGISTERS
        PUSH    BX
          ETC...
```

Listing 4.

Debugging drivers

OEMs have ICEs and Logic Analyzers, but what can the rest of us do? The first time I tried to test a homegrown device driver, it crashed the system (not surprising). It came as quite a shock when I realized that I didn't have access to all that test equipment at home. (Not to mention the engineer to twiddle that last button to get it working for me). Rather than spend megabucks on buying or leasing high-powered test equipment, I decided to try to find software ways to debug drivers on a working MS-DOS system.

The first trick I used was to declare my *strategy* and *request* routines as far *PUBLIC*s in the driver. This allowed me to link the driver into a test program and call the *strategy* and *INIT* routines as *FAR* routines and single step through them with *DEBUG*. Single stepping allows you to check to make sure the jump table in your request routine is functioning properly.

After that, I would build a fake *request* header in the main test program and watch what the *INIT* routine did to it. A LINK map/MAP would tell me, for example, if the break address was returned correctly. I declare these entry points *PUBLIC* in all my drivers now; you can leave the *PUBLIC* statements in with no effect in the driver later. When you are satisfied that it works in this test environment, try loading it for real again. In my past experience writing device drivers for minicomputers like DEC PDP11s and VAXs, I have learned an important fact: if you can get your device driver to load without crashing the system, you're halfway there.

Since the *INIT* routine is the first part of a driver that I write, I'm often impatient to see something happen. So I use another debugging trick that gets me on-line sooner: I write a *NUL* device driver first. What this means is after I write the *INIT* routine, I write a dummy routine for all the rest of the *request* routines that don't do anything.

If each of these routines returns the correct status, you can load your drivers and send data to them with no errors! Also with no visible sign that anything works, but not getting crashes or errors is an invisible sign that you did something right. From this point on, you can write each of the *request* routines one at a time and debug them fully before proceeding. Alternately, you can write a minimum routine for each request and add all the bells and whistles later.

When things are not going well, it is handy to know exactly how far a driver got before crashing. To get a visual indication of this, I used the IBM PC graphics card as a debug output device. Because this card is memory mapped, it is easy to make characters appear on it from within a driver. I wrote the macro in Listing 5 to do this in as painless a way as possible.

Then I liberally sprinkled invocations of this macro in all parts of my driver: in the *strategy* routine (*NOTIFY S*), in the *request* routine (*NOTIFY R*), in the *INIT* routine (*NOTIFY I*), in the *read* routine (*NOTIFY* r), etc. When my driver was running, characters would march across the top of the screen, and I would know how far the driver got before it failed.

If that did not narrow the problem down enough, I would include more invocations of *NOTIFY*, rebuild, and test again. On systems without memory mapped screens, other ways could be found to get a single byte of data out of the system. A UART could send the byte to an ASCII terminal, or you could beep the systems bell in code if no other technique presents itself.

And finally, I did find ways of using *DEBUG* to single step through a live driver, although this method is dangerous. The first problem with using *DEBUG* on a driver is figuring out where in memory your driver was loaded.

Now a MS-DOS programmer doesn't know or care where programs land in memory, (Rule No. 2: Never write to known memory locations), but I'm willing to bend the rule just for the driver debug period. (Besides, we are doing it in a driver). My drivers have all been interrupt driven, so the quick and easy way to find this type of driver is to look at where the interrupt vector points.

For simplicity in writing the drivers, I always copy the driver CS register into the vector segment, then the offset to the interrupt service routine into the vector offset. This means that the interrupt service routines share the same CS as the rest of the driver and can access the same subroutines and data. Coincidentally, it also means that the vector segment points at the first paragraph of the driver itself and I can use that to look at the live driver with *DEBUG*.

If your driver is not interrupt driven, you will have to find other ways to find it. My first suggestion would be to have your *INIT* routine load a dummy interrupt service routine address into an unused vector in low memory. Then you can pretend that it's an interrupt driven device for a second and look up the vector segment value as described previously.

Armed with that information, you can check up on your driver by simply examining, and even changing, its data structures. But if you are like me, you want to put breakpoints in the debugger and catch it in the act—even single step through it. This is possible, but you have to be very cautious.

Basically, the problem is that MS-DOS seems to have only one request block that it uses to talk to all devices. The system uses it to call the device, but when a breakpoint happens, the system re-uses that same request block to print out the registers for *DEBUG*. If your driver has not looked at the request block yet, that's too bad—the data in it is gone.

But if you can place the breakpoint past the spot where the driver reads the request block or if you can substitute the correct values, then you can use *DEBUG*—even to single step through a live driver on a working MS-DOS system. One thing worries me though: what happens when you allow the driver to continue and return to the calling routine? Doesn't the driver return a status code to the wrong program? Does that hurt? And what happens if you type Q to *DEBUG* in an interrupt service routine? Doesn't the system stack get permanently bent out of shape?

Worrying about the answers to the questions has me re-booting every time I use this technique. This requires a little planning in advance. You get all the information you need ready, you decide exactly where you are going to place the breakpoint and how far to single step afterwards. You write a test program to call the driver with just the right request. And then you take one shot at it. If you find the

```
;MACRO FOR INDICATING STATUS OF DRIVERS ON AN IBM
           ;COLOR ADAPTOR CARD.

DEBUG    DD      0B800000H       ;ADDRESS OF COLOR SCREEN,
                                        ;USE 0B000000H FOR
NOTIFY  MACRO   CHAR                                  MONO.
        PUSH    ES              ;SAVE ANY REGISTERS
        PUSH    DI              ;THAT YOU USE
        PUSH    AX
        MOV     AX,'&CHAR'      ;ARGUMENT IS ASCII CHAR
        LES     DI,CS:DEBUG     ;GET ADDRESS OF SCREEN
        STOS    BYTE PTR [DI]   ;CHAR APPEARS ON SCREEN
        INC     DI              ;INCREMENT TO NEXT CHAR
        MOV     WORD PTR CS:DEBUG,DI      ;STORE OFFSET BACK
        POP     AX              ;RESTORE ALL REGISTERS
        POP     DI
        POP     ES
        ENDM
```

Listing 5.

VEDIT PLUS

Easy to Use Word Processor with the Power of BASIC

VEDIT has been acclaimed for the last four years the industry standard in text editing.

InfoWorld
Software Report Card
VEDIT 1.36

	Poor	Fair	Good	Excellent
Performance	☐	☐	☐	☑
Documentation	☐	☐	☐	☑
Ease of Use	☐	☐	☑	☐
Error Handling	☐	☐	☐	☑

Now there's VEDIT PLUS.

VEDIT PLUS is easy to learn, yet can do things far beyond ordinary word processors:

- Sort a mailing list
- Perform arithmetic computations
- Compare files
- Edit mulitple files of unlimited size

The power of VEDIT PLUS is equal to a high-level language - it gives you:

- If-then-else decision making
- Pattern matching
- User prompts and input
- An optional Z80 to 8086 translator

Expect a lot from VEDIT PLUS. It's from CompuView - nationally recognized for user support.

For a demo disk, more information or the dealer nearest you call toll free:

1-800-327-5895

VEDIT PLUS is available for practically every microcomputer . . .$225

CompuView
PRODUCTS, INC.

1955 Pauline, Ann Arbor, Mi 48103
(313) 996-1299 • (800) 327-5895

CIRCLE 81 ON READER SERVICE CARD

bug, you re-boot so you have a clean system to edit the fix with. If you don't find the bug, re-boot anyway so the system doesn't get confused and trash that hard disk.

Sample drivers

I have heard rumors that there are bugs in the sample drivers included in the back of the device drivers chapter of the IBM DOS 2.0 manual. I have definitely been warned that the sample drivers in the MS-DOS OEM distribution kit will not work if assembled. When I started writing my ASYNC driver for the IBM COM ports, I also started writing this article. I planned on including the driver listing as an example at the end of the paper.

Unfortunately, I got carried away and included everything in this driver that I have ever wanted in a terminal driver on a personal computer. The listing is 35 pages long, and I assumed that *COMPUTER LANGUAGE* people would not have the space to spare. So the source to the async driver, file name ASYNC.ASM, will be available for downloading on the *COMPUTER LANGUAGE* Bulletin Board Service and on the magazine's account on CompuServe (type "GO CLM"). Also on-line will be a file called TERM.C, which is the C source to a program that sets all the set-able features of the async device driver on an MS-DOS system. Or, if you don't have access to a modem to call the BBS or *COMPUTER LANGUAGE*'s account on CompuServe, you can write directly to me (Mike Higgins, The Computer Entomologist, P.O. Box 197, Duncans Mills, Calif. 95430). I'm going to ask for a small fee of $10.00 to cover media, postage, and handling hassles. (That's really not an out-of-line price, have you ever tried to buy "free" software from the U.S. government?) But I'm offering the code itself as a completely public domain piece of software with no restrictions.

I boot my system straight onto the ASYNC driver now by including a *CTTY ASYNC2* command in my AUTOEXEC batch file. I have tried removing my color adapter card and running the system with an external terminal and keyboard only. It works great! This article was almost completely typed in through the ASYNC driver and printed to an RS-232 printer for all the rough drafts. I'm reasonably sure that all the bugs are out of it.

The following is a list of the features I have included in this driver to help me get full use out of my RS-232 ports:

- A 128-byte interrupt driven buffer is on both input and output
- Supports XON/XOFF protocol or hardware data throttling on output
- Bells echo when the in-buffer overflows
- Control-C and control-S are passed straight through on nondestructive reads. (This solves a problem on many MS-DOS systems—programs cannot be aborted when the input ring buffer already has characters in it)
- Several escape sequences are scanned for on input to perform special features:

Flush the input buffer
Redraw the last line
Delete the current line
Skip output to fast forward
Reboot the system (like ALT-SHIFT-DEL)

- Intelligent half duplex is supported on input. (Intelligent means the characters are not echoed until MS-DOS reads them out of the input buffer)
- Errors can be reported to MS-DOS in a variety of ways or not at all
- Parity can be stripped off of characters on input or output
- Characters can also be converted to upper case on input and/or output
- All these features, plus the baud rate, can be read or changed with an *IOCTL* call to the driver
- A sample C program that sets/clears the preceding features and reports on the current driver status is included with the driver.

A device driver is an important and powerful feature of MS-DOS. Using drivers will allow hardware manufacturers to incorporate new devices into MS-DOS in a system in a uniform manner. Writing virtual device drivers could make it possible for software manufacturers to isolate device specific portions of their code and transport programs to a wider variety of MS-DOS systems.

I believe that the MS-DOS users would be well-served if both hardware and software manufacturers paid more attention to the possibilities inherent in loadable device drivers. In this article, I have clarified some of the things I found confusing when I first attempted to write an MS-DOS driver. I've also given examples of some important parts of drivers and communicated some debugging tools that I found extremely useful. Armed with this information, I hope that more people, both manufacturers and home users, will find it easier to use this feature of MS-DOS.

Mike Higgins works for Time Arts, a computer graphics company in Santa Rosa, Calif. He received a B.A. in computer science from the Univ. of Calif., Berkeley, Calif., in 1975 and has been a professional programmer for about 13 years.

(LISP) FOR A.I.

UO-LISP Programming Environment The Powerful Implementation of LISP for MICRO COMPUTERS

LEARN LISP System (LLS.1)
(see description below *$39.95*

UO-LISP Programming Environment Base Line System (BLS.1) *$49.95*

Includes: Interpreter, Compiler, Structure Editor, Extended Numbers, Trace, Pretty Print, various Utilities, and Manual with Usage Examples. (BLS.1) expands to support full system and products described below.

UO-LISP Programming Environment: The Usual LISP Interpreter Functions, Data Types and Extensions, Structure & Screen Editors, Compiler, Optimizer, LISP & Assembly Code Intermixing, Compiled Code Library Loader, I/O Support, Macros, Debug Tools, Sort & Merge, On-Line Help, Other Utility Packages, Hardware and Operating System Access, Session Freeze and Restart, Manual with Examples expands to over 350 pages. Other UO-LISP products include: LISPTEX text formatter, LITTLE META translator writing system, RLISP high level language, NLARGE algebra system. Prices vary with configurations beyond (BLS.1) please *send for FREE catalog.*

LEARN LISP System (LLS.1): Complete with LISP Tutorial Guide, Editor Tutorial Guide, System Manual with Examples, Full LISP Interpreter, On-Line Help and other Utilities. LEARN LISP fundamentals and programming techniques rapidly and effectively. This system does not permit expansion to include the compiler and other products listed above.

LISP Tutorial Support (LTS.1): Includes LISP and Structure Editor Tutorial Guides, On-line Help, and History Loop. This option adds a valuable learning tool to the UO-LISP Programming Environment (BLS.1). *Order (LTS.1) for* **$19.95.**

REQUIRES: UO-LISP Products run on most Z80 computers with CP/M, TRSDOS or TRSDOS compatible operating systems. The 8086 version available soon.

TO ORDER: Send Name, Address, Phone No., Computer Type, Disk Format Type, Package Price, 6.5% Tax (CA residents only), Ship & Handle fee of $3.00 inside U.S. & CN, $10 outside U.S., Check, Money Order, VISA and MasterCard accepted. With Credit Card include exp. date. Other configurations and products are ordered thru our *FREE catalog.*

Northwest Computer Algorithms
P.O. Box 90995, Long Beach, CA 90809 (213) 426-1893

CIRCLE 46 ON READER SERVICE CARD

RP/M T.M.

By the author of Hayden's "CP/M Revealed."

New resident console processor RCP and new resident disk operating system RDOS replace CCP and BDOS without TPA size change.

User 0 files common to all users; user number visible in system prompt; file first extent size and user assignment displayed by DIR; cross-drive command file search; paged TYPE display with selectable page size. SUBMIT runs on any drive with multiple command files conditionally invoked by CALL. Automatic disk flaw processing isolates unuseable sectors. For high capacity disk systems RDOS can provide instantaneous directory access and delete redundant nondismountable disk logins. RPMPIP utility copies files, optionally prompts for confirmation during copy-all, compares files, archives large files to multiple floppy disks. RPMGEN and GETRPM self-install RP/M on any computer currently running CP/M®2.2. Source program assembly listings of RCP and RDOS appear in the RP/M user's manual.

RP/M manual with RPMGEN.COM and GETRPM.COM plus our RPMPIP.COM and other RP/M utilities on 8" SSSD $75. Shipping $5 ($10 nonUS). MC,VISA.

118 SW First St. - Box G
Warrenton, OR. 97146

Micro Methods, Inc.

(503) 861-1765

CIRCLE 52 ON READER SERVICE CARD

QUALITY SOFTWARE AT REASONABLE PRICES

CP/M Software by
Poor Person Software

Keyed Sequential Files **$39.95**
CP/M 2.2 Extension. Available to all languages with a BDOS interface. Variable length Records to 508 bytes, single key to 32 bytes, files to 256k bytes. Source for "C" language support library.

Poor Person's Menus **$29.95**
Interactive menu design and edit program with "C" language source for runtime support library. Package includes "C" source for an interactive Mailing List Manager using Poor Person's Menus and separately purchased Keyed Sequential Files (compatible with Mailing Label Printer). Write your own interactive data manager programs.

Poor Person's Spooler **$49.95**
All the function of a hardware print buffer at a fraction of the cost. Keyboard control. Spools and prints simultaneously.

Poor Person's Spelling Checker **$29.95**
Simple and fast! 33,000 word dictionary. Checks any CP/M text file.

Mailing Label Printer **$29.95**
Select and print labels in many formats.

Window System **$29.95**
Application control of independent virtual screens.

All products require 48k CP/M 2.2 and are available on 8" and 5" Northstar formats, other 5" formats add $5 handling charge. California residents include sales tax.

Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

CIRCLE 26 ON READER SERVICE CARD

68000

C, PASCAL, FORTRAN

CROSS AND NATIVE SOFTWARE DEVELOPMENT TOOLS

OASYS is both a developer and distributor of the finest, full featured, reliable, high quality set of **68000, 8086** and **NS 32000** tools in the world.

We offer:

- **COMPILERS (Native and Cross)**
  C, Pascal, FORTRAN, PLM-68K
  (COBOL-68K, BASIC, PL-1 coming)
- **Cross and native Macro Assembler Linkers, Object Module Librarians and Cross Reference tools.**
- **C Symbolic (source level) Debuggers**
- **Simulator with disassembler**
- **Floating point math package**
- **Safe C™ checkout compiler**
- **Clue™ C line count utility**
- **Up/down line load comm. utilities**

Available on: VAX/VMS, Bsd 4.1, 4.2, III, V; Prime/Primos; 68000's; 8086's; and dozens more.

OASYS
60 Aberdeen Ave.
Cambridge, MA 02138
(617) 491-4180

68K CPF-11/84

CIRCLE 24 ON READER SERVICE CARD

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES

- **Programmer's Referral List**
- **Compare Products**
- **Help find a Publisher**
- **Evaluation Literature free**
- **BULLETIN BOARD - 7 PM to 7 AM 617-826-4086**
- **Dealer's Inquire**
- **Newsletter**
- **Rush Order**
- **Over 500 products**

"Guaranteeing" Software – Possible?

It is a very messy issue. Trust is required; but trust means that the publisher, retailer or user gets left open to losing, being hurt. The Programmer's Shop trusts customers and **guarantees SOLID VALUE.** Ask about it. The next "*Programmer's Letter*" offers ideas.

RECENT DISCOVERIES

ProYAM Communications Package - everything a programmer would want. TTY, VT 100, 3101, MODEM 7, BBS, Remote, macros, windows. MSDOS. $139.

ARTIFICIAL INTELLIGENCE

EXSYS - Expert System building tool. Full RAM, Probability, Why, Intriguing, serious. PCDOS $295

GC LISP - "COMMON LISP", Help, tutorial, co-routines, compiled functions, thorough. PCDOS $475

IQ LISP - MACLISP & INTERLISP. Full RAM. Liked. PCDOS $155.

TLC LISP - "LISP-machine"-like, all RAM, classes, turtle graphics 8087 for CP/M-86, MSDOS $235

TLC LOGO - fast, classes. CPM $139

PROLOG-86 - Learn fast, Standard, tutorials, samples of Natural Language, Exp. Sys. MSDOS $125

Expert System front-ends for PROLOG: APES ($275), ES/P ($1895)

Other solid alternatives include: MuLISP-86 ($250), WALTZ LISP for CPM ($159), MicroPROLOG ($275).

EDITORS FOR PROGRAMMING

BRIEF Programmer's Editor - undo, windows, reconfigurable, macro programs, powerful. PCDOS $195

VEDIT - well liked, macros, buffers, CPM-80-86, MSDOS, PCDOS $119.

MACINTOSH

We evaluate, carry every available programmers product. Ask.

C LANGUAGE

INSTANT C - Interactive development - Edit, Source Debug, run. Edit to Run - 3 Secs. MSDOS $500

"INTRODUCING C" - Interactive C to learn fast. 500 page tutorial, examples, graphics. PCDOS $95

MEGAMAX C - native Macintosh has fast compile, tight code, K&R, toolkit, .OBJ, DisASM MAC $295

CROSS COMPILERS by Lattice, CI. VAX to 8086. VMS $3000

C LIBRARIES

COMMUNICATIONS by Greenleaf ($149) or Software Horizons ($139) includes Modem7, interrupts, etc. Source. Ask for Greenleaf demo.

C SHARP Realtime Toolkit - well supported, thorough, portable, objects, state sys. Source $600

PORTABLE C-LIB: Same calls for IBM, IIe, CP/M, C64, more. Screen, I/O, Graphic, more. $125.

ROMPack - special $Main .EXE editor, source, tech support, 8086. $195

DEBUGGERS

PERISCOPE DEBUGGER - load after "bombs", symbolic, "Reset box", 2 Screen, own 16K. $295

SOURCE PROBE by Atron for Lattice, MS C, Pascal. Windows single step, 2 screen, log file. $395

FORTRAN LANGUAGE

RM/FORTRAN - Full '77, big arrays, 8087, debugging, xref. MSDOS $525

DR/Fortran-77 - full ANSI 77, 8087, overlay, full RAM, big arrays, complex NUMS., CPM86, MSDOS. $249

OTHER LANGUAGES

ASSEMBLER - ask about Microsoft MASM-86 ($125) improvements or its new competitors.

"BASICA COMPILER": Better BASIC all RAM, modules, structure. $185

HS/FORTH - '79 & '83 Standards, full RAM, ASM, BIOS, interrupts, graph, multi-task, optimizer MSDOS $250.

MBP COBOL has screen control, strong doc, '74 interm., fast. MSDOS $695.

SUPPORT PRODUCTS

BASIC DEVELOPMENT SYSTEM - (BDS) for BASICA; Adds Renum, crossref, compress . . . $115.

PLINK-86 for Overlays, most lang., segment control. MSDOS $345

PROFILER-86 - faster programs with less work. Learn quick, symbolic, All Lang. MSDOS $125

SCIL - Source Librarian to manage Versions, Doc, Minimize disk space, confusion. MSDOS $335

"C" LANGUAGE

	OUR PRICE
MSDOS: C86-8087, reliable	call
Instant C - Inter., fast, full	500
Lattice 2.1 - improved	call
Microsoft C 2.x	329
Williams, debugger, fast	call
CPM80: Ecosoft C-now solid, full	225
BDS C - solid value	125
MACINTOSH: Softworks	365
Megamax-object, full	295
Consulair's MAC C	295

Compare, evaluate, consider other Cs

BASIC

	RUNS ON	
Active Trace-debug	86/80	75
BASCOM-86 - MicroSoft	8086	279
BASIC Dev't System	PCDOS	115
BetterBASIC - 640K	PCDOS	185
CB-86 - DRI	CPM86	439
Prof. BASIC Compiler	PCDOS	89
Databurst - screens	MSDOS	215
SCREEN SCULPTOR	PCDOS	119

Ask about ISAM, other addons for BASIC

SERVICE

ADVICE - We help choose a product to fit your work. Even if you don't know of a product, we'll help.

EDITORS Programming

	RUNS ON	OUR PRICE
BRIEF - Intuitive, flexible	PCDOS	195
C Screen with source	86/80	75
Epsilon - like EMACS	PCDOS	195
FINAL WORD-for manuals	86/80	215
MINCE-like EMACS	PC/80	149
PMATE-powerful	8086	195
VEDIT-full, liked	86/80	119

UNIX PC

COHERENT - for "C" users	PClike	475
COHERENT-NCI-Realtime	PClike	call
XENIX - plus C to MSDOS	PC	1295

Ask about run-times, applications, DOS compatibility, other alternatives. UNIX is a trademark of Bell Labs

LANGUAGE LIBRARIES

GRAPHICS: GraphiC-source in C	MSDOS	195
GRAPHMATIC-3D. FTN, PAS	PCDOS	125
HALO-fast, full-all lang.	PCDOS	175
FILE MGNT: BTrieve-all lang.	MSDOS	215
CIndex + -source, no royal.	86/80	375
CTree-source, no royal.	ALL	375
dbC ISAM by Lattice	8086	250
db VISTA-"Network" Structure	MSDOS	465
PHACT-up under UNIX, addons	MSDOS	250
OTHER: CUtil by Essential	MSDOS	139
Greenleaf - 200 +	MSDOS	165
CSharp - **Real-Time**	MSDOS	600
PORTABLE C to PC, Mac, II	Many	125
SOFT Horizons - Blocks I	PCDOS	139
SCREEN: CURSES by Lattice	PCDOS	125
CView - input, validate	PCDOS	195
MetaWINDOW - icons, clip	PCDOS	139
PANEL - many lang, term	MSDOS	265

Call for a catalog, literature, and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™

128-L Rockland Street, Hanover, MA 02339.

Visa Mass: 800-442-8070 or 617-826-7531 MasterCard

8517

FORTRAN

	RUNS ON	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS	$ 239
DR Fortran-86 - full '77'	8086	249
PolyFORTRAN-XREF, Xtract	PCDOS	165

OTHER PRODUCTS

Assembler & Tools - DRI	8086	159
Atron Debugger for Lattice	PCDOS	395
cEnglish - dBase C	MSDOS	795
C Helper: DIFF, xref, more	86/80	135
CODESMITH-86 - debug	PCDOS	139
MacASM-full, fast, tools	MAC	100
MBP Cobol-86 - fast	8086	695
METAWINDOW - graph, fonts, clip	PCDOS	135
Micro: SubMATH-FORTRAN full	86/80	245
Microsoft MASM-86	MSDOS	85
MSD Debugger	PCDOS	119
Multilink - Multitasking	PCDOS	265
PC/FORTH + -well liked	MSDOS	219
PFIX-86 Debugger	MSDOS	169
PL/1-86	8086	495
Polylibrarian - thorough	MSDOS	95
PolyMAKE	PCDOS	95
PROFILER - flexible	MSDOS	125
Prolog-86-Learn, Experiment	MSDOS	125
SLK/F - Copy Protection	PCDOS	150
SYMD debugger-symbols	PCDOS	119
TRACE86 debugger ASM	MSDOS	115

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and **POs.** All formats available.

CIRCLE 94 ON READER SERVICE CARD

Part II: Debugging

Programming Macros in C

By Alexander B. Abacus

Debugging is considered to be an activity planned for in advance during program design and extended into program maintenance. In Part II of my series, "Programming Macros in C," I will describe macros that help in planning, writing, and maintaining debugging code for programs written in C.

Next month we'll finish off the series with a look at how to use macros to translate a common representation of a table of data into different but related tables required by different programs.

During program development, programmers insert into the code statements that are helpful for debugging but must not be executed in the final, production version of the program. Good programmers plan for debugging and treat debugging statements as a permanent part of the program; they write them before the first debugging run and do not remove them after debugging is completed.

Writing debugging statements in advance speeds up debugging. Leaving debugging statements as a permanent part of the program speeds up debugging of future program modifications. When the macros for C described in this article are used together, they will make writing and maintaining debugging code easier. All the listings presented in this article can be obtained in electronic form by calling the *COMPUTER LANGUAGE* Bulletin Board Service or account on CompuServe (type "GO CLM"). On both media, the files will be labeled DEBUG.*.

DEBUGIN() and DEBUGOUT()

If debugging statements are left permanently in the program, it is essential that the program can be compiled with or without generating debugging statements. Generation of debugging statements should be controlled by compiler options without changing the source code. Conditional compilation can be achieved in C using preprocessor commands. If we decide to generate debugging code only when preprocessor symbol Debug is defined, then debugging code must be within the *#if/#endif* construct:

```
#if defined Debug
    /* debugging code */
#endif
```

Preprocessor symbol Debug is not defined in the source code but can be defined using the -D option of the C compiler:

```
$ cc -D Debug program.c
```

Alternatively, we can define a macro *DEBUGIN()* that will generate code given by its argument when the symbol Debug is defined and ignore its argument otherwise. This macro and its opposite, *DEBUGOUT()*, are defined as in Listing 1.

Now debugging code can be written as:

```
DEBUGIN
(
    /* debugging code */
)
```

We choose to use the second method because it offers several advantages.

Preprocessor commands have a rigid format. They must be on separate lines and some compilers require that # is the first character on the line (with no white space before it). This disrupts the indentation of the source code intended to show the structure of the program.

If the debugging code to be generated is short, with preprocessor commands it must take at least three lines. With the macro it can be written on one line as:

```
DEBUGIN( /* debugging code */ )
```

```
#if ! defined Debug
#  define DEBUGIN(Statements) /* empty */
#  define DEBUGOUT(Statements) Statements
#else defined Debug
#  define DEBUGIN(Statements) Statements
#  define DEBUGOUT(Statements) /* empty */
#endif ? defined Debug
```

Listing 1.

Having these two simple macros defined makes it easier to define other more complex macros that depend on the same preprocessor symbol Debug. A good example of this is the macro *StdMsg()* defined later.

Are there any restrictions on the arguments passed to *DEBUGIN()* and *DEBUGOUT()*? The argument of these macros may be text of any complexity, limited only by the preprocessor restriction on the size of the macro argument string.

All parentheses within the text must be paired. An unpaired right parenthesis will terminate the argument list of the macro. An unpaired left parenthesis will prevent proper termination of the argument list of the macro. All commas must be either within quotes or within pairs of parentheses or the preprocessor will interpret them as macro argument separators.

When debugging statements span multiple lines and the C compiler issues an error message for one of those statements, the line number in the error message usually refers to the first or the last line of such a group, not to the line containing the error. In extreme cases it may be necessary to take the statements out of the macro context to get the exact line number in the error message. This also applies to the macro *DEBUG()* described next.

Some C compilers do not allow macro calls to cross line boundaries. For such compilers, macro name, opening parenthesis, arguments, and closed parentheses must all be on the same line. Unlike macro definitions, lines cannot be continued even by using a backslash as the last character on the line.

DEBUG()

When the debugging statements are generated it is also desirable to control their execution by run-time switches. Values of the switches will usually be set from the command line. Before we write a macro to test the switches, let us consider how to set them from the command line. That will help us chose the data representation for switches.

If we were satisfied to have only one debugging switch with two possible values, namely ON or OFF, a key-letter flag on the command line will be enough to set the switch, as in:

```
$ command -! arguments
```

If we insist on multiple switches, we must have a way to identify them. If we identify them by integers we can set them as follows:

```
$ command -!1,4,17 arguments ...
```

If we chose to identify them by single characters instead of integers, the command line becomes easier to parse:

```
$ command -!adq arguments ...
```

If we increment the corresponding switch for each letter in the argument string, we can have switches with more than two values. The command:

```
$ command -!aaaddq arguments ...
```

will set the switch 'a' to 3, switch 'd' to 2, and switch 'q' to 1. Let us also agree that:

```
$ command -! arguments ...
```

```
#define START(ProcId, ProcName, ProcForm) \
   DEBUGIN \
   ( \
     struct \
     { \
       char* procName; \
       char  procId; \
       char* procForm; \
     } thisProc; \
     struct \
     { \
       char blockLevel; \
     } thisBlock; \
     thisProc.procName = ProcName; \
     thisProc.procId = ProcId; \
     thisProc.procForm = ProcForm; \
     thisBlock.blockLevel = 0; \
     if ( debug[0] || debug[thisProc.procId] ) \
     { \
       StdMsg ( stderr , "Procedure entered.\n" ); \
     } \
   ) /* DEBUGIN */

Example:

   int fun()
   {
     int result;
     START('f',"fun","%d")
     /* ... */
     RETURN(result)
   }
```

Listing 2.

is shorthand for setting all switches to a non-zero value.

The choice of one-character identifiers for switches conveniently limits the number of switches to the size of the character set. Assuming an ASCII character set, we can define the switches as:

```
char debug[128];
```

and refer to a switch 'a' as in

```
#if defined Debug
     if (debug['a'] >= 3)
     {
           /* debugging statements */
     }
#endif defined Debug
```

When using macro *DEBUG()* this can be condensed into:

```
DEBUG
( 'a', 3,
     /* debugging statements */
)
```

Macro *DEBUG()* is defined as:

```
#define DEBUG(Id,Level,Statements) \
  DEBUGIN( if (debug[Id] >= Level) \
    { Statements } )
```

The three arguments are defined as follows:

- Argument *Id* is used as an index into the array of switches
- Argument *Level* is the lowest value of the switch for which generated statements are executed
- Argument *Statements* are any executable C statements including macro calls that generate executable statements.

In the final implementation of this macro (given in the file debug.h) the first parameter is dropped. The index of the switch to be used by *DEBUG()* is specified as the first argument of the macro *START()* that must be coded at the beginning of any function that uses *DEBUG()*. This index is saved in an automatic variable that replaces the parameter *Id* in the definition of *DEBUG()*.

START()

Macro *DEBUG()* allows us to code any statements we wish executed during debugging at any place in the program. Most of the time debugging statements will consist only of printing values of important variables at those places in the program where their values are modified. During debugging we always want to know which portions of the code have been executed. To make obtaining that information easier we may want to provide macros for all control flow mechanisms (*if, while, for* . . .) and have them print out the outcome of their decisions. We do it here for the most important control flow mechanism: function invocation and return.

For debugging trace of function execution we provide two macros: *START()* and *RETURN()*. *START()* will be coded at the beginning of each function, once per function. *RETURN()* will be coded instead of the return keyword.

Macro *START()*, presented in Listing 2, must be used to start any function that uses macros *DEBUG()*, *StdMsg()*, *RETURN()*, *EXIT()*, and *ABEND()*. It prints a trace message about entering the function. It also saves the values of the arguments of this macro call. This information is saved in the automatic variables local to the function, therefore it is kept separately for different functions and for recursive invocations of the same function.

Because this macro generates both declarative and executable statements, it must be coded as the first executable statement in the function. C requires that all declarations within a block precede all executable statements of the block.

This macro takes three arguments.

The first argument must be a character constant. The value of this argument is used as the index of the switch in the calls of the macro *DEBUG()* within the function.

The second argument is a string constant and must supply the name of the function. Macro *StdMsg()* uses it as part of the message prefix.

The third argument is a string constant and must supply formatting specification for returned value. It is used by macro *RETURN()* to print the value being returned by the function. Formatting specification is passed to *fprintf()* as the second argument (format). If the function does not return a value, or to suppress printing of the returned value, code this argument as 0. Otherwise follow the rules for the format argument of *fprintf()*.

StdMsg()

It is useful to prefix debugging and error messages with information about the exact place in the program where the mes-

```
#define _ ,
#define StdMsg \
   (void) fprintf(stderr \
   , DEBUGIN("%14s|%4d|%8s|%c%d|") DEBUGOUT("%14s|%4d|") \
   , __FILE__ \
   , __LINE__ \
     DEBUGIN \
     ( \
     _ thisProc.procName \
     _ thisProc.procId \
     _ thisBlock.blockLevel \
     ) /* DEBUGIN */ \
   ); \
   (void) fprintf /* (stderr, ... ) */
```

Listing 3.

sage is issued. We can accomplish that by using a macro instead of the function call:

```
fprintf( stderr, ... );
```

If we keep the syntax of our macro identical to the syntax of the function call for *fprintf()*, we can define the macro as presented in Listing 3.

Without macros *DEBUGIN()* and *DEBUGOUT()* we would have had to write separately two complete macro definitions. To avoid the restriction on commas within the arguments of the macro *DEBUGIN()*, we had to define a symbol that will be translated to a comma by the preprocessor. Here we have used the preprocessor symbol consisting of a single character _ (underscore).

The syntax of the macro *StdMsg()* is (apparently) the same as the syntax of the function call for *fprintf()*:

```
StdMsg( stderr, ... );
```

Note that the first argument must be *stderr*, because two *fprintf()* calls are used. The first call prints identifying information to a standard error stream. The second call prints arguments to the stream specified as the first argument. There are two reasons for requiring this first argument:

- To keep the syntax of this macro identical to the syntax of the function *fprintf()*.
- Macros cannot have a variable number of arguments. Macro *StdMsg()* is then implemented as a macro without parameters, i.e., *StdMsg*, and the parentheses are just part of the original source text following the macro call, thus forcing the requirement that all arguments of the function *fprintf()* must be retained.

When symbol Debug is not defined, the identifying information consists of:

- Name of the source file, supplied by the preprocessor defined symbol __FILE__
- Line number in the source file, supplied by the preprocessor defined symbol __LINE__.

When symbol Debug is defined, the following additional identifying information is provided:

the macro *START()*, which must be used to start any function using *StdMsg()*

the macro *START()*, which must be used to start any function using StdMsg()

- Index of the debugging switch for the function
- Smallest value of the switch for which the line is printed.

The last two items assist in selecting the values of debugging switches to reduce the volume of debugging output.

```
#if ! defined Debug
#  define Nothing
#  define Void void
#else defined Debug
#  define Nothing 0
#  define Void int
#endif ? defined Debug

#define RETURN(Value) \
   DEBUGIN \
   ( \
     if ( debug[0] ) \
     { \
       StdMsg ( stderr , "Procedure returned" ); \
       if ( (thisProc.procForm != 0) && ((*thisProc.procForm) != '\0') ) \
       { \
         (void) fprintf ( stderr, " value: "); \
         (void) fprintf ( stderr, thisProc.procForm , Value); \
       } \
       (void) fprintf ( stderr, ".\n"); \
     } \
   ) /* DEBUGIN */ \
   return Value ;
```

Listing 4.

RETURN(), EXIT(), and ABEND()

To have a complete trace of function execution, all functions should start with macro *START()* and finish their execution with one of the following macros: *RETURN()*, *EXIT()*, or *ABEND()*.

Macro *EXIT()* should be used for a normal return to the operating system. The argument should be return code zero. Macro *ABEND()* should be used for abnormal program termination. The argument should be non-zero return code.

Macro *RETURN()*, presented in Listing 4, should be used for a normal return to the caller. This macro prints the value returned by the function using the format saved by the macro *START()*. If the function returns no result, this macro must be coded as:

```
RETURN(Nothing);
```

Symbol *Nothing* is defined as 0 during debugging and as empty otherwise. If we coded:

```
RETURN();
```

substitution of *Value* in the argument list of *fprintf()* will yield:

```
(void) fprintf ( stderr, this-
        Proc.procForm , );
```

producing a syntax error. This also implies that functions defined as *void* must be defined as *int* during debugging. Therefore they should be declared and defined as *Void* instead of *void*. Symbol *Void* is defined appropriately.

ASSERT()

The argument to this macro is a condition that must be satisfied. If it is not, the text of the condition is printed and the program is abnormally terminated. This macro (Listing 5) uses previously described macros *DEBUGIN()*, *StdMsg()*, and *ABEND()*.

Source files

All differences between the code that must be generated for the source file containing the function *main()* and source files containing other functions are localized in file debug.h. Symbol *Storage* must be defined in the source file containing *main()* before including the file debug.h.

Macros *Extern* and *Init()* are to be used within include files containing statements destined to become definitions or declarations of global data, depending on where they are included. Let a variable be defined within an include as

```
Extern char * words[ ] Init( "one" _
        "two" _ "three" );
```

```
#define ASSERT( Predicate ) \
   DEBUGIN \
   ( \
     if ( ! (Predicate) ) \
     { \
       StdMsg \
       ( stderr \
       , "False assertion: (%s).\n" \
       , "Predicate" \
       ); \
       ABEND(1); \
     } \
   ) /* DEBUGIN */
```

Listing 5.

C

Software Development
PCDOS/MSDOS

Complete C Compiler
- Full C per K&R
- Inline 8087 or Assembler Floating Point, Auto Select of 8087
- Full 1Mb Addressing for Code or Data
- Transcendental Functions
- ROMable Code
- Register Variables
- Supports Inline Assembler Code

MSDOS 1.1/2.0 Library Support
- All functions from K&R
- All DOS 2.0 Functions
- Auto Select of 1.1 or 2.0
- Program Chaining Using Exec
- Environment Available to Main

c-window™ Symbolic Debugger
- Source Code Display
- Variable Display & Alteration Using C Expressions
- Automatic Commands
- Multiple Breakpoints by Function & Line Number

8088/8086 Assembler
- FAST — Up to 4 times Faster than IBM Assembler
- Standard Intel Mnemonics
- Compatible with MSDOS Linker
- Supports Full Memory Model

8088 Software Development Package $199.00

Includes: C Compiler/Library, c-window, and Assembler, plus Source Code for c-systems Print Utility

c-systems

P.O. Box 3253
Fullerton, CA 92634
714-637-5362

CIRCLE 16 ON READER SERVICE CARD

Included within the main file this will generate the definition with initial value:

```
char * words[] = { "one" , "two" ,
     "three" };
```

Included within other files, the same line will generate only a declaration of an external variable:

```
extern char * words[];
```

Note that within the argument of the macro *Init()* an underscore is used to separate initial values instead of a comma. This is because a comma would be taken for the separator of macro arguments. The symbol _ (underscore) is defined to generate the comma required to separate items in a list of initial values.

The debug.h file will behave gracefully if included more than once in a source file. It may be necessary to include this file more than once if the value of the preprocessor symbol Storage changes. All symbols dependent on Storage are undefined and then defined again without preprocessor warnings.

An implementation

Now, let's take a look at a genuine implementation of the concepts we've discussed so far, illustrated in Listing 6. This code illustrates the use of macros *START()*, *DEBUG()*, *StdMsg()*, *RETURN()*, and *EXIT()*. It also illustrates how to keep the test driver in the same source file with the function to be tested. The symbol Driver must be defined on the compiler command line when the test driver is to be compiled.

Those interested in exploring even further the subject of debugging macros in C may be interested in getting the rather lengthy original file referred to in this article called DEBUG.H, which I've placed on the *COMPUTER LANGUAGE* BBS and account on CompuServe.

Alexander Abacus has a B.S. in electrical engineering and is a software consultant for CGA Computer Inc., Cranford, N.J. His software experience includes work for Sperry, IBM, and AT&T Bell Laboratories.

```
/* strext.c: @(#) (re)allocate memory and append a string */
/* by Alexander B. Abacus */

#ifdef COMMENT
/*
/* PURPOSE:
/* Append a string to another string dynamically allocating
/* additional storage.  The first pointer must be either 0,
/* when the second string is copied into dynamically allocated
/* storage, or must be the pointer returned by malloc() or
/* this function strext(). The second pointer must not be 0.
/*
/* EXAMPLE:
/*   string = (char*) 0;
/*   string = strext(string, "Head...");
/*   string = strext(string, "Middle...");
/*   string = strext(string, "Tail.");
*/
#endif COMMENT

#include "debug.h"

extern void      free();
extern char*     malloc();
extern char*     realloc();

#include <string.h>

char* strext( head, tail )
char* head;
char* tail;
{
  char* newString;
  START('a',"strext","(string) \"%s\"")

  DEBUG ( 1, StdMsg ( stderr , "Tail is \"%s\".\n" , tail ); )

  if ( head == (char*) 0 )
```

Listing 6 *(Continued on following page).*

Illustration: Anne Doering

```
      {
        newString = malloc( (unsigned) strlen(tail) + 1 );
        if ( newString == (char*) 0 )
        {
          DEBUG ( 1, StdMsg(stderr, " Memory allocation failed.\n"); )
          RETURN( (char*) 0 );
        }
        (*newString) = '\0';
      }
      else
      {
        DEBUG ( 1, StdMsg(stderr, "Head is \"%s\".\n", head); )
      newString = realloc( head, (unsigned) strlen(head) + strlen(tail) + 1 );
      if ( newString == (char*) 0 )
      {
        DEBUG ( 1, StdMsg(stderr, " Memory reallocation failed.\n"); )
        RETURN( (char*) 0 );
      }
    }
  newString = strcat(newString, tail);
  DEBUG ( 1, StdMsg(stderr, "New string is \"%s\".\n", newString); )
  RETURN (newString);
} /* strext() */

/* ----------------------------------------------------------------- */

#if defined Driver

#define Storage static
#include "debug.h"

Void main (argc, argv)
int argc;
char* argv[];
{
  int    index;
  char*  string;
  START('m',"main","")
  DEBUGIN ( debug['a'] = 1; )

  string = (char*) 0;
  for ( (index = 1); (index < argc); (++index) )
  {
    string = strext(string, argv[index]);
  }

  EXIT (0);
} /* main */

#endif defined Driver ---------------------------------------------

/* strext.c: End of file. */
```

Listing 6 *(Continued from preceding page).*

MODULA-2 PROGRAMMING TOOLS

A collection of utility modules ready to link into your programs and greatly speed programming efforts and the operation of programs.

Each tool is supplied as a definition module with in-line documentation, an implementation module with full source code and a ready-to-link object module. A fully-linked ready-to-run test program with source code is included.

Each module is implemented using Logitech's Modula-2/86^{tm}, Version 1.1 and MS-DOS/PC-DOS^{tm} Version 2.0 or later unless otherwise specified. All modules are upward compatible with Microsoft's Xenix^{tm} operating system as specified in the Microsoft MS-DOS Programmer's Reference Manual.

MemUtils: high-speed memory utilities coded using 8086 string instructions.
Keyboard: a complete IBM-PC keyboard handler.
ScreenOps: high-speed routines for controlling IBM-PC text screen. Based on ROM BIOS calls.
FileOps: direct access to MS-DOS file handling functions via DOS function calls.
DirOps: direct access to MS-DOS's hierarchial directories via DOS function calls.
DiskUtils: miscellaneous disk and drive utilities via MS-DOS function calls.
SingVD: calculates singular values of real-values matricies.
MicroMouse: direct access to all 16 Microsoft Mouse funtcions via mouse system software function calls.

Memutils	**$29**	All three for **$59**	Entire package of 8 modules - all with source code and test programs for **$189**
Keyboard	**$39**		
ScreenOps	**$39**		
FileOps	**$39**	All three for **$79**	
DirOps	**$39**		
DiskUtils	**$29**		
MicroMouse	**$49**		
SingVD	**$89**		

Developed by: Thomas H. Woteki, Ph.D.

Add $3/order shipping and handling   VA residents add 4% sales tax

Call **703/ 522-8898** or send your order to: **Information Systems Incorporated**
1901 No. Fort Myer Drive, Arlington, VA 22209

-Quality Software At Low Prices-   **-Save Time With Expert Tools-**

CIRCLE 20 ON READER SERVICE CARD

Another in a series of productivity notes on UNIX™ software from UniPress.

COMPILERS

AMSTERDAM COMPILER KIT

Subject: A complete Kit of compilers, cross compilers and assemblers.

The Amsterdam Compiler Kit is the only C and Pascal UNIX package which includes a wide range of native and cross tools. The Kit is also easily modifiable to support custom targets.

Features:

- *C and Pascal compilers (native and cross) for UNIX machines.*
- *Host and target machines include VAX™ 4.1/4.2 BSD, PDP™-11/V7, MC68000™ and 8086™ Cross assemblers provided for 8080,™ Z80,™ Z8000,™ 8086,™ 6800,™ 6809,™ 68000,™ 6502 and PDP-11.*
- *The Kit contains complete sources* of all programs, plus comprehensive internals documentation on how to make modifications needed to add a new program language or new target machine.*

**A source UNIX or C license is required from AT&T.*

Price:

Full Source System	*$9950*
Educational Institutions	*995*

Selected binaries are available – contact us with your machine type.

For more information on these and other UNIX software products, call or write: UniPress Software, Inc., 2025 Lincoln Hwy., Edison, NJ 08817. Telephone: (201) 985-8000. Order Desk: (800) 222-0550 (Outside NJ). Telex: 709418. Japanese Distributor: SofTec 0480 (85) 6565. European Distributor: Modulator SA (031) 59 22 22

OEM terms available.
Mastercard/Visa accepted.

UNIX is a trademark of AT&T Bell Laboratories. VAX & PDP 11 are trademarks of Digital Equipment Corp. MC68000; 6800 & 6809 are trademarks of Motorola Corp. 8080 & 8086 are trademarks of Intel Corp. Z80 & Z8000 are trademarks of Zilog, Inc.

UniPress Software
Your Leading Source for UNIX™ Software.

CIRCLE 40 ON READER SERVICE CARD

Hardware special feature

Trends in Multiprocessing Systems

By Sol Libes

The application of 32-bit microprocessors in desktop workstations is a natural progression from the previous generations of 8- and 16-bit microprocessor-based systems. Multiuser and multiprocessing systems using the 8- and 16-bit devices have become quite widespread, particularly with the availability of 16-bit devices.

Although there are multi-mpu 8-bit systems, such as the TurboDos-based systems (see August 1984 issue of *Microsystems*), these have been systems in which each user had his/her own dedicated CPU which happened to be housed in common cabinet-sharing facilities, such as a hard disk and printer. And there are multiprocessing systems based on a single microprocessor (e.g., Digital Research's Concurrent DOS and MP/M).

Newer system architectures are beginning to come to market which take advantage of the decreasing prices of microprocessors and memory. These systems employ multiple processors to form a resource pool orchestrated by either a common operating system or a number of communicating operating systems to service the dynamically changing processing load. Most of the activity in this area is related to systems used for on-line business-transaction processing applications. Fault tolerance [1] is also a key feature of most of these systems. One manufacturer claims that using hardware techniques, no single hardware fault can bring the system down and that the system will recover automatically from a fault within milliseconds.

The purpose of this article is to take a look at this newly emerging and fast-growing segment of the microprocessor-based computer marketplace.

Multiuser systems

A transaction-processing system is characterized by a large number of users trying to access and update a data base, often a large data base. An airline reservation system is an example of a very large transaction-processing system. Users are working in real time and demand real-time response.

Typically, the demand on such systems has grown quickly and in a short time exceeded the capacity of the machine. Often a transaction processor that is adequate today is no longer adequate six months or a year from now. Thus expansibility is a very important feature for transaction-processing systems.

Traditional single-CPU multiuser minicomputer systems have architectural bottlenecks. They bottleneck on the back end due to limited disk-channel capability and bottleneck on the front end due to limited communications processing. And they bottleneck in between because their single CPU is handling all the applications as well.

Thus manufacturers early on began offloading front-end processing to separate computers systems, such as the IBM 3705 which provides dedicated front-end processing. Some also attach back-end processors (e.g., data base processors) to offload transaction-processing tasks.

These systems are thus very expensive, not easily altered to meet changing conditions and generally lack any hardware fault-tolerance features. A multicomputer system requires multiple copies of the operating system even if they are associated with multiple job or application processors. A TurboDos system, for example, is really a multi-computer system with single-board computers in a common box sharing peripheral I/O devices such as a disk system and printer. Such a system is therefore considered to be a loosely-coupled, tightly-packaged (all processors in one box) system.

Another variation is the single-application processor with multiple I/O processors. A tightly-coupled system is one with several processors sharing memory with one copy of the operating system. The multiple processors can serve as application, print, file, I/O and graphics processors. Thus all resources are available at all times for all tasks equally.

Tightly-coupled systems come in two varieties—those with limited upgrades to perhaps four or five processors and those with large upgrades to 10, 20 or perhaps hundreds of processors. The large shared-memory systems suffer from one drawback. They tend to have a very high entry price because to solve the problem of so many processors having to access a single memory, extra hardware has to be put into the system.

Another approach is to combine the loosely- and tightly-coupled systems to obtain the benefits of both approaches. Such a system combines multi-clusters where each cluster has tightly-coupled multiprocessors. One can add clusters to the system or add processors to a cluster to increase processing power. Transaction-processing multiprocessor systems provide very high performance, can be easily reconfigured, can be easily and economically upgraded in a modular fashion, and can have fault-tolerant features.

Multiprocessors are not perfect for all environments. Running a single very large application program may mean that only one processor is running the program and the other processors are sitting idle. On the other hand, in a transaction environment that consists of lots of little jobs, multiprocessors are ideal. A single-unit processor would tend to bog down when it has to manage a large number of tasks. In the multiprocessor environment a few processes can be assigned to each processor, reducing the overhead of a single-unit processor. Thus multiprocessors are much more efficient.

Hardware fault tolerance

With a single-unit processor, if you have a hardware failure, your entire system is out until it is repaired. With a multiprocessor FT^2 system, when a hardware fault occurs in one processor the system can be reconfigured, omitting the failed processor—thus restoring at least partial processing capability.

If there are redundant buses, redundant disks and mirrored disks, then full availability of the system can usually be restored by merely switching the peripherals to a processor that has not failed. Software can be added to the system to make the switch-over automatic and entirely transparent to the user. Such a system is generally referred to as a fully fault-tolerant system.

With the cost of processors coming down, the likelihood is that in the future all multiprocessor systems will include some level of fault-tolerance.

The ideal computer system

The ideal system would allow users to have a variable amount of memory and a variable number of processors. Thus one user on the system may be using multiple processors while another is using only part of a processor. The system must be able to easily accommodate change since environments are always undergoing change.

All job processors in the system would share equally in the processing tasks. Any or all could execute the operating system, even simultaneously. The processors would thus become a pool of available resources. The system would have dynamic load balancing—the ability for any process to execute on any processor at any time with the distribution of processes being handled automatically, dynamically and transparently.

The system would dynamically adjust to a smaller number of processors in the case of a processor failure. And as more processors are added the system would automatically adjust to the size of the processor pool. An ideal system would allow incremental expansion in multiple dimensions. Thus the designer could vary the number of processors, the amount of memory, the number and types of I/O channels, and the number and types of local area networks with no software or hardware redesign or re-education for the users.

The ideal system would also be a true transparent multiprocessing system with distributed job-management software to provide dynamic load balancing, job scheduling and resource sharing. It would employ standard internal busing (e.g., Multibus, S-100, etc.), standard interfaces to disk drives (e.g., SMD) to I/O devices (e.g., RS-232, SDLC, HDLC, bisync) and local-area networks (e.g., Ethernet). It would be based on a standard 32-bit microprocessor engine (e.g., Motorola 68020 or National Semiconductor series 32000) with memory management and hardware floating-point arithmetic units.

Which 32-bit micro to use?

With the introduction of 32-bit microprocessors, all of the approaches to multiprocessing will undergo dramatic shifts in performance.

Manufacturers who have already introduced true multiprocessor systems have based them on the Motorola 68000 and 68010 16-bit devices. They have selected these devices because of the increased processing power over the Intel 8086. Further, Motorola had indicated early on that their 32-bit device, the 68020, would be upward object-code compatible from the 68000 and 68010.

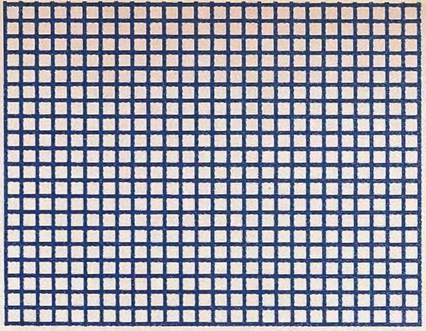
Since the systems manufacturers, who already have 680X0 systems out, expect to move up to 32-bit devices for the multiprocessor systems, the object-code compatibility becomes a significant plus. On the other hand, new start-up manufacturers just entering the multiprocessing market do not have to worry about upward compatibility and appear to be going for the National Semiconductor 32000 family of devices.

National Semiconductor and Motorola have already begun sampling their 32-bit microprocessors and Intel is expected to start sampling its 80386 by the middle of '85. NCR, Hewlett Packard and AT&T are already in production on 32-bit microprocessor chip sets (which are used in their new 3b2 series of multiuser UNIX-based systems), but these are restricted to their internal use. Although there are rumors that AT&T may begin selling its 32-bit chips in the not-too-far-distant future.

In addition to enhanced internal architecture and addressing, the 32-bitters also run faster. They are routinely clocked at well over 12 Mhz—about the same clock rates used in supercomputer mainframes such as the Cray 1 and Cyber 205. However, it should be noted that support ICs which run at these clock speeds are not expected to become available until some time after the microprocessor chips.

The availability of 256K RAM chips means that the 32-bit CPUs will routinely come with base memory sizes of 1MB expanding upward to 16MB and more. Motorola has already announced that it will use a new 84-pin package on a new version of the 68020 that will provide 30 address lines so that the microprocessor can directly address up to one gigabyte.

Intel was actually the first integrated-circuit manufacturer to introduce a 32-bit microprocessor. In early 1980 Intel shocked the industry with the announce-



ment that it had developed the iAPX-432, a full 32-bit micro with 32-bit I/O as well as an internal 32-bit architecture. The unit departed from the traditional architecture used by previous 8-bit and 16-bit products. The device was actually a three-chip set, designed to utilize the Ada language in a multiprocessor system architecture. The chip set was thus intended to provide higher performance in object-oriented knowledge-based systems that Intel believed would be the architecture of the future.

It is interesting to note that the 432 family of devices provide fault-tolerance features via functional-redundancy checking. Self-checking can be enabled via an external signal.

By mid-1981, Intel was supplying samples of the 432 chip set to OEMs, and several companies began system-development efforts based on the chip set. However, it soon became apparent that the performance of the device left something to be desired. Initial samples, operated at 4 Mhz clock speeds, proved slower than Intel's 8086 and Motorola 68000 16-bit systems in simple standard benchmarks. Also, the new departure in architecture meant system designers were starting at ground zero and needed a great deal of retraining and support. At the time Intel was still learning about the device, and support was very limited. The result was that by early '82 virtually all the OEMs had stopped their 432 system-development projects.

Intel, realizing that it was having big problems with its 432, decided to undertake the development of another 32-bit micro, to be called the 80386, which promises upward compatibility with the 8086 family. Intel has yet to release any data on the device and is promising samples in mid '85. Thus Intel, who at one time was first out of the gate, has dropped to last out, and the device appears to be about a year behind Motorola and National[3].

Zilog has announced that it will introduce a 32-bit micro chip, to be called the Z80,000, with sampling to begin this fall.

Another instruction set?

Do real men create their own instruction sets or do they use off-the-shelf architecture? Today the answer is that rather than look forward to a three- to four-year development effort with no guarantee of success, they use off-the-shelf processors.

Today's highly complex VLSI and very efficient 16- and 32-bit microprocessors have eliminated the need for system designers to develop their own instruction sets—which was the situation that existed with most mini and mainframe designers. The marketplace now requires that companies, including giants such as IBM, utilize off-the-shelf microprocessors to get their systems to market more quickly. The systems will be designed to meet specific marketing niches and include hardware and software features on top of the off-the-shelf products.

Another operating system?

Do real men create their own operating systems or do they use a standard operating system? To create a new operating system today involves all the risk of poor quality and extensive testing and development of new software development tools and application programs.

Therefore today few OEMs want to take the risk of using a new operating system and would rather go with a popular and proven multiprocessing operating system such as UNIX. This is really the basis for the increasing popularity of UNIX today. In other words, the selection of UNIX as the operating system for multiprocessing systems is basically a marketing decision.

UNIX wasn't designed to be a distributed-processing, transaction-oriented, real-time system. Manufacturers have had to make major modifications to the UNIX kernel while still maintaining the standard kernel interface to accommodate their particular architectures and to provide file-system reliability for real-time communications handling and real-time distributed processing. These were not trivial tasks and represented a major effort for all companies using UNIX.

Many companies had to, in effect, rewrite the entire kernel. However, since the deficiencies, fixes and improvements of UNIX have been well documented at conferences and in the literature, the companies felt it was better to modify or rewrite UNIX rather than undertake a lengthy development effort and introduce a product and then find out what its deficiencies were. The result, they felt, was a reliable and robust operating system.

Vendors have also had to make substantial modifications to UNIX to achieve the distributed multiprocessing and fault-tolerant operation. In addition, substantial modifications to the externals of UNIX were required to make it more user-friendly and better fit different application markets than what UNIX was originally designed for.

Suppliers catering to the technical and scientific marketplace appear to be selecting the Berkeley UNIX version 4.2, while suppliers aimed at the business-transaction marketplace appear to be selecting AT&T's UNIX System V implementation. UNIX will obviously in the future evolve into something quite different from what it is today. There is a question as to whether the external interfaces will be stable so that application programs can operate on a wide variety of UNIX implementations.

Some companies are planning to provide both versions of UNIX with a user-defined switch to select either 4.2 or System V kernel. We can also expect to see versions of UNIX that offer a superset of 4.2 and System V, so that no switch is required.

Although the new multiprocessor systems are intended to support environments consisting of multiusers running a lot of small jobs, there is some expectation that these systems may be used by a single user running a single job that is divided among multiple concurrent processors. Thus it may be possible to have an application program use multiple processors to increase computing power for a given task running in real time. A UNIX pipe, for example, might be used to divide the multiple processes among multiple processors. There are already plans by several vendors to develop concurrent versions of development tools. Thus, a concurrent version of the C language could have a process automatically split among several processors.

Some actual systems

A few multiprocessor systems utilizing 32-bit microprocessors have already been announced and should be reaching the marketplace shortly.[4]

Convergent Technologies, which primarily makes systems for other companies (its largest customer is AT&T) recently announced its MegaFrame, which it classifies as "an advanced low-cost, time-sharing supermini."

The MegaFrame runs up to 36 processors inter-connected via an 88-megabit per sec data bus. Thus the system consists of tightly-coupled stand-alone computers. Each processor, with its own memory and operating system, is on its own board and runs independently. As more processors are plugged into the system the performance increases.

Each processor is dedicated to handling specific tasks. The heart of the system is the applications processor using the new Motorola 68020 32-bit mpu. Each processor concurrently runs a demand-paged virtual-memory version of UNIX System V. Processors communicate via a real-time message-based architecture. The file system, data-base handling and all file-oriented transactions are running on a separate back-end DBMS processor. All file-oriented transaction processing is off-loaded from the application processor to the file-handling processor.

There are two types of front-end processors—the cluster processor and terminal processors that run real-time UNIX operating systems—which get around some of the inherent real-time limitations of UNIX. The cluster processor talks to the local-area network (at 1.8 megabits per sec) while the terminal processors talk to RS-232 type terminals.

A minimum system contains three processors—a file processor, a cluster or terminal processor and an application processor and a 50MB hard disk. A 64-user configuration would contain eight processors, external SMD drives and 8MB of RAM. The MegaFrame is designed for large-user, low-cost applications and thus does not have any fault tolerance features.

Auragen Systems has announced a multiprocessor system that is higher in price but offers fault-tolerance and several other advantages. It utilizes a multi-cluster architecture with each cluster consisting of several tightly-coupled processors sharing a common memory. Each cluster has an executive processor with a 68000 and 256K of local RAM. The executive processor runs the kernel of the Auragen operating system, which is a message-based version of UNIX. It is responsible for all interprocessor message traffic and also for coordinating various processors within a cluster.

The application processor has a virtual-memory management unit and two 68010 processors, each of which runs independent programs residing in main memory. Main memory may have up to 8MB.

Further, there can be up to four communications processors in a cluster, each of which has a 68000, up to 256K of local RAM and supports various protocols (e.g., HDLC, bysync, SDLC). There can be up to four file processors, each of which supports up to four SMD disk drives and up to four tape drives. A cluster by itself is a powerful multiprocessor. Up to 32 clusters can be added together via dual ported 16MB system buses. Hence, if one cluster fails, the terminals and peripherals can be accessed by another cluster and dual-porting approach to provide a high level of fault tolerance.

The system is glued together with a message-based version of UNIX. It is fully distributed and transparent to the user. The user accesses the system as if it were a single-processor system and is unaware of which cluster or which processor is running the application at any given time.

How Do You Measure A Good C Compiler?

CODE SPEED & SIZE
The Lattice C Compiler "generates code that is quite compact and fast running." *Peter Norton, PC Magazine*

CONSISTENT RELIABILITY
"The Lattice Compiler has performed reliably and predictably." *R. Phraner, Byte Magazine*

COMPILE TIME
"Lattice is a real performer." *Houston, Brodrick, Kent, Byte Magazine*

THIRD-PARTY LIBRARIES
More than 40 library products are currently available for Lattice

UNIX V COMPATIBILITY
The Lattice Library is UNIX V-compatible

DEBUGGER SUPPORT
The Lattice C-SPRITE Debugger is now available

DOCUMENTATION
Lattice "is thorough and excellent." *D. Clapp, PC Magazine*

UPDATE POLICIES
Lattice provides free bugfix updates for 90 days

COOPERATING PRODUCTS
New LMK Utility, dBC Library, CVUE Screen Text Editor, CURSES Screen Library and GSS Graphics are available from Lattice

VENDOR REPUTATION
Lattice is used in more commercial products than any other C Compiler. No run-time license is required.

ALL MEMORY MODELS
Lattice C has 7 memory models available to allow the best solution for the task at hand

AVAILABILITY OF CROSS COMPILERS

SATISFACTION GUARANTEED!

Ask About Our "Trade Up To Lattice C" Policy

LATTICE®

Lattice®, Inc.
P.O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950
TWX 910-291-2190

UNIX is a trademark of
AT&T Bell Laboratories

CIRCLE 53 ON READER SERVICE CARD

The message-based version of UNIX allows the building of user-transparent fault tolerance. The application programs do not require modification to run on the fault-tolerant system. Auragen has enhanced its version of UNIX to work in a commercial-transaction environment with features such as data integrity, data security and a user-friendly interface.

A three-cluster system with four processors per cluster would contain 12 processors. Auragen claims that since the 68010 benchmarks at well over 1MIP, such a system would have a performance in the 10-15 MIP range—which is enormous by today's standards. Auragen has disclosed that it is working 32-bit microprocessor versions of its system.

The company Sequoia Systems takes another approach to providing a high-speed multiprocessor system with a very high degree of fault tolerance. The system is divided into segments. Each segment has a dual 40MB bus and handles up to eight modules. The system can be expanded up to 24 segments. For example, one segment may have eight processor modules and another have eight memory and I/O modules.

Processor modules are based on the 68010 microprocessor with a 128K cache, high-speed bus interfaces and numerous performance-enhancing hardware. The memory modules are 2MB modules with over 100 test and set locks to support resource sharing in a multiprocessor environment. The I/O processors are 68000-based with over a half MB of RAM. The large buffer storage provides very high I/O performance.

The company claims that the segmented bus structure significantly improves performance, which consists of dual 40MB buses. The problem with conventional buses is that as you add more electrical loads the throughput, or bandwidth, of the buses decreases. This invariably happens at precisely the time when you need that extra capacity because you have added more modules. The buses, which operate at 40MB per sec have been segmented in such a way that no one segment sees more than nine electrical loads. The inner segment bus can see as many as 24 loads, but it is a physically extended bus which looks very much like a transmission line to support that many loads. Up to 24 segments can be added to have as many as a 216 bus.

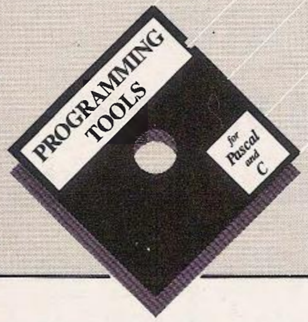
Hardware-based fault tolerance

Faults can occur any time and, therefore, to have a truly fault-tolerant system, it is necessary to know when those faults occur and to monitor them at all times. If software is used to monitor the faults, there isn't time to do anything else, so this is a good application for hardware.

To achieve what Sequoia calls true fault tolerance, it uses a hardware-based system. By using duplication and comparison, extensive use of coding, protocol monitors, etc., the company claims to provide instantaneous detection of any fault. Software is used to recover from a fault. Sequoia claims the ability to recover from most faults occurs within a few hundred millseconds. It claims that there is no dedicated redundancy for recovering from faults and that all modules in the system are always productively used. Sequoia also uses the UNIX-based operating system.

Flexible Computer Corp. has announced plans to introduce a computer system that will utilize as many as 10 National Semiconductor 32032 32-bit microprocessors, each on a separate single-board computer (SBC) plugged into a 20-slot mother board. Each SBC will contain the 32032, a math coprocessor, 1MB of RAM and a cache memory. Thus each SBC can handle separate concurrent tasks. Flexible expects to furnish UNIX System V with the system.

Sequant Computer, a recent Intel spin-



Software Tools for Serial Devices

Are you developing a C or Pascal program that will include functionality like this?

- ***File transfer between computers***
- ***Realtime data acquisition***
- ***Printer/plotter control***

Blaise Computing Inc. has a set of tools for you. ASYNCH MANAGER™ is a library of software routines, including all source code, which will dramatically reduce the amount of time you need to get your applications working.

Using ASYNCH MANAGER,™ you can control any hardware device employing the RS-232 interface on an IBM PC or operationally compatible micro computer. Moreover, your PC will be able to operate at fast speeds, since interrupt-driven routines and flexible buffer sizes are featured.

As with other packages in our Programmer Productivity Series, there are no royalties for incorporating our routines into your systems. Other high-level language interfaces are planned. Contact us directly to order or to obtain additional specifications.

ASYNCH MANAGER™ **$175.00** includes the first language interface, second language **$75.00** additional.

Other products to speed your development projects in C and Pascal include: **TOOLS**™ comprises a library of routines providing advanced string handling, forms utilities, screen handling and more. Features a general BIOS gate—**$125**

TOOLS 2™ is a second library of routines focusing on DOS 2.0+ operating systems capabilities. Perform memory allocation, program chaining, file and buffer handling from within your C or Pascal program—**$100**

VIEW MANAGER™ is our screen development system which dramatically reduces programming required for screen displays. Block mode data collection, verification, overlays are all featured—**$275**. Source Code for routine library—**$150**

BLAISE COMPUTING INC.
2034 Blake Street Berkeley, CA 94704
(415) 540-5441

CIRCLE 8 ON READER SERVICE CARD

off, has as yet not announced a product. However, the company is expected shortly to announce a multiprocessor system based on the National Semiconductor 32032 microprocessor and some proprietary silicon hardware. Its system is expected to come closer to achieving the ideal system described earlier, and this is certainly a company worth watching.

Computer Consoles sells an on-line transaction-processing system that features up to eight 68000 CPUs, 68000-based disk and terminal controllers, and special inter-computer controllers (ICCs). The system has fault-tolerant features. It uses a UNIX-based operating system with one copy per CPU. Each CPU can have up to 4MB with error checking and correction and battery backup. Performance of 5.6 MIPS is claimed.

The ICCs monitor CPU tasks via a time-out scheme and if a stalled CPU is detected the task is passed to the next available CPU. Bad ICCs are detected by a CPU voting scheme. ■

Footnotes

1. It is interesting to note that it was NASA who pioneered fault-tolerant systems to ensure reliability of systems in space.
2. For an in-depth discussion of FT techniques and the architectures used on several of the current FT systems, I recommend reading the August 1984 issue of *Computer* magazine. The entire issue is devoted to the subject. *Computer* magazine is published by the IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, Calif. 90720, (503) 283-7433.
3. It is premature to count the Intel 432 microprocessor out of the 32-bit micro market. There is no doubt that its radically new architecture has proved difficult to work with, and the early production devices proved disappointingly slow in performance. However, Intel has continued to work on the device and has introduced a new version of the device with improved performance. Further, Intel has worked hard at developing the software-development support needed by computer system designers. However, Intel is effectively no longer marketing the device. Now comes word that Siemens AG has joined with Intel in a venture to enhance the 432 with the objective of using the devices in multiprocessor configurations expected to provide performance equal to the DEC VAX 11/780, a superminicomputer, to be followed by a system in the VAX 11/750 performance class (the 750 is DEC's new enhanced version of the 780). The goal is to introduce this device in late '86 or early '87. The project is expected to provide some compatibility with the 8088/8086 architecture and eventually a version of the 432 for use in desktop machines.
4. These descriptions are based on manufacturers' supplied information rather than firsthand knowledge.

Suppliers mentioned in this article:

1. Convergent Technologies, 2441 Mission College Blvd., Santa Clara, Calif. 95050, (408) 727-8830
2. Computer Consoles Inc., 97 Humboldt St., Rochester, N.Y. 14609, (716) 482-5000
3. Flexible Computer Corp., 1801 Royal Lane, #210, Dallas, Texas 75229, (214) 869-1234
4. Auragen Systems Corp., 2 Executive Dr., Fort Lee, N.J. 07024, (201) 461-3400
5. Sequoia Systems, 3 Metropolitan Corp. Center, Marlboro, Mass. 01752, (617) 480-0800
6. Sequant Computer, 14360 NW Science Park Dr., Portland, Ore. (503) 626-5700

Sol Libes founded and was the editor of Microsystems, *one of the first microcomputer publications. (See* COMPUTER LANGUAGE*'s interview with Libes in the January issue, pp. 23-25). Libes teaches digital electronics and microcomputer at Union County College in New Jersey.*

Realia COBOL.
Numbers speak louder than words.

Compilation Speed (minutes:seconds)

Lines in Program	Realia COBOL	mbp COBOL	Level II COBOL	R-M COBOL	Microsoft COBOL
1,000	:51	8:33	3:42	5:05	5:11
5,000	3:30	48:07	16:58	*	45:26

*Could not successfully compile the program.

Execution Time Ratio
(Gibson Mix; calculated S-Profile)

Realia COBOL	mbp COBOL	Level II COBOL	R-M COBOL	Microsoft COBOL
1.0	3.6	14.7	21.6	22.3

Sieve of Eratosthenes

0.818 seconds per iteration

All benchmark tests were performed on an IBM PC-XT with 192KB of memory. IBM PC-XT is a registered trademark of International Business Machines Corporation; mbp COBOL, of mbp Software and System Technology; Level II COBOL, of Micro Focus; R-M COBOL, of Ryan-McFarland; and Microsoft COBOL, of Microsoft.

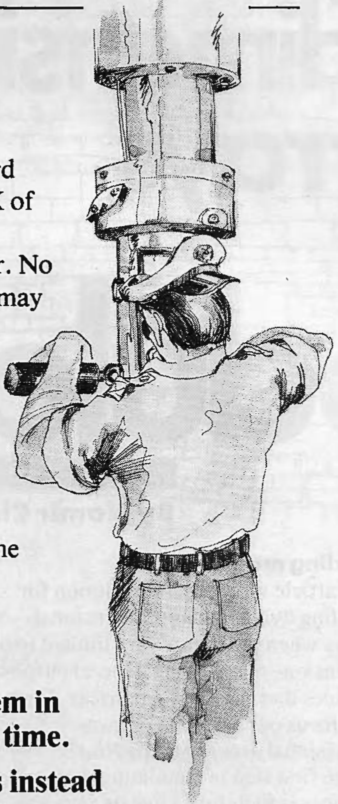
+ IBM VS COBOL compatibility.

REALIA inc.

$995

10 South Riverside Plaza
Chicago, Illinois 60606
(312) 346-0642

CIRCLE 76 ON READER SERVICE CARD



Debugging Bugging You?

Torpedo program crashes and debugging delays with debugging dynamite for the IBM PC ...

UP PERISCOPE!

First, you install the hardware.

The hardware's a special memory board that fits in a PC expansion slot. Its 16K of write-protected memory contains Periscope's resident symbolic debugger. No runaway program, however berserk it may be, can touch this memory!

Then you UP PERISCOPE.

Use Periscope's push-button break-out switch to interrupt a running program ... even when the system's hung! Periscope supports Assembly, BASIC, C and Pascal. In addition to the usual debugging capabilities, some of Periscope's features are:

Stop your system in its tracks at any time.

Use symbol names instead of addresses.

Run a program on one monitor and debug on another.

Monitor your program's execution with Periscope's comprehensive breakpoints.

Debug memory-resident programs.

Put your time to better use.

The Periscope system is $295. It carries a 30-day money-back guarantee and includes the memory board, remote break-out switch, debugger software, 100-page manual, and quick-reference card. The memory board is warranted for one year. A demonstration disk is $5.00.

System requirements for Periscope are an IBM PC, XT, AT or Compaq, PC-DOS, 64K RAM, 1 disk drive and an 80-column monitor. For MasterCard and Visa orders only, call 800/421-5300 (ext. R96) 24 hours a day. For additional information, call 404/256-3860 from 9 AM to 5 PM Eastern Time.

Get your programs up and running;

UP PERISCOPE!

Data Base Decisions / 14 Bonnie Lane /Atlanta, GA 30328

CIRCLE 30 ON READER SERVICE CARD

ACTIVE TRACE

"Software that lives up to its promises. *When a Basic program doesn't work the way you want it to, this package. . . will help you track the problem down. . . Scope is a tool for the beginning, advanced, or professional programmer, and it begins where the cross reference maps leave off."*

Howard Glosser, *Softalk for the IBM Personal Computer* July '84, pp 120-121

***"Extremely useful program. . .** Anyone doing much programming in Basic should appreciate Active Trace a lot."*

Jerry Pournelle, *Byte Magazine* April '83, p 234

***"A marvelous Basic programming aid. . .** It's just amazing to watch a program you wrote run under Scope, and debugging becomes if not trivial, then at least doable".*

Thomas Bonoma, *Microcomputing,* Dec. '83, p 22

*"**. . . a really neat utility. . .** designed to untangle even the most convoluted Basic program. . . . The documentation is almost worth the price of the package."*

Susan Glinert-Cole, *Creative Computing,* July '84, p 210

Active Trace will lead you through your program letting you know variable values (all variables or just those you specify) as they change. Your program's internal activity is presented on your screen, or printer, or it can be saved on disk. **It's simple, effective and works with the BASIC you already own.**

Active Trace $79.95

Includes Scope, XREF mapping and documentation

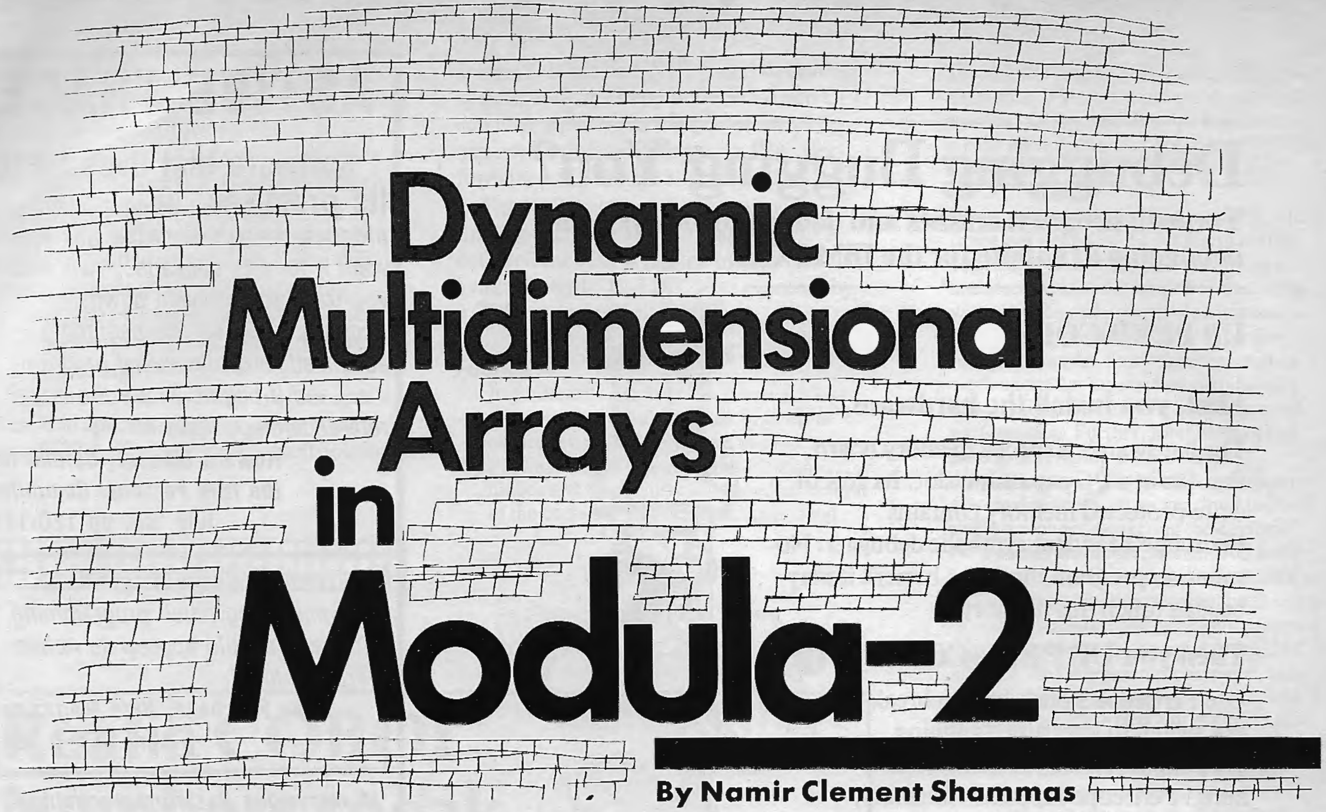
Active Trace is available for most **MS-DOS** and **CPM 2.2** systems and supports the special features of Brand specific versions of Microsoft Basic such as **Basica** on the **IBM-PC**.

AWARECO
Active Software

P.O. Box 695 Gualala, CA 95445
(707) 884-4019
800-358-9120(US) 800-862-4948(CA)

Active trace, Active software, and Scope are trademarks of AWARECO—CPM is a trademark of Digital Research—MS-DOS and Microsoft are trademarks of Microsoft Corporation—IBM-PC is a trademark of IBM Corp.

CIRCLE 3 ON READER SERVICE CARD



Dynamic Multidimensional Arrays in Modula-2

By Namir Clement Shammas

The advent of the Modula-2 programming language reflects the growing trend toward more efficient large-project software development and updating. One of this language's more powerful features is that it allows the development of general purpose library modules that are compiled separately.

Modula-2's strict type checking, inherited from Pascal, poses some limitations. But these restrictions have been partially alleviated because Modula-2 allows for dynamic (open) arrays to be declared in a procedure argument, as shown in the following procedure, *Sum*, which returns the sum of the array *X*,

```
PROCEDURE Sum(X : ARRAY OF
    REAL) : REAL;
```

Procedures such as this one allow for processing arrays of any size as long as the basic data type is consistent (e.g., *REAL* in this example). It is possible to write procedures that handle dynamic, one-dimensional arrays of any basic or user-defined type.

In Modula-2, general purpose library modules can be developed to handle these arrays. This is indeed an improvement over Pascal's rigidity. But, even with many of Modula-2's enhancements, a problem still exists for dynamic multidimensional arrays because the procedures are limited to one dimension.

Tackling matrices

This article will present a solution for handling dynamic multidimensional arrays when procedures are limited to one dimension—developing general purpose modules that can tackle matrices. First let's focus our attention on two-dimensional arrays of type *REAL*.

The first step in simulating dynamic matrices is to unfold a matrix into one long array. Adopting the convention that the matrix columns are "chained," we can write an equation that maps the two-dimensional matrix elements onto the new one-dimensional array, with a lower dimension index of one:

```
Index in array = (Column_index - 1)
      * Max_Rows + Row_Index
```

This equation shows that the array index does not depend on the number of columns. This technique allows for visualizing matrices as special cases of one-dimensional arrays. With the latter we can use the open array feature.

Implementation code for a matrix management module for handling floating point matrixes, called MatMan, can be seen in Listing 1. (If you would like to download an electronic copy of this listing, it is available on the *COMPUTER LANGUAGE* Bulletin Board Service as well as on our CompuServe account.) Observe that the first four procedures are used similarly to map matrices onto an array. The following is an explanation for what the procedures actually do (the procedures are grouped in two sets):

- Procedures *Loc* and *Loc0* are called when a user program or procedure is handling dynamic matrices with an equal and fixed number of rows. The identifier HIROW (which is exported by the module) is assigned the number of rows. The two procedures are identical except that the index value returned by *Loc0* is 1 less than that of *Loc*. The same offset exists between the lower dimension bound of a dynamic matrix (equal to 0) and that of a matrix-containing array (assigned as 1). If the lower dimension bound of the matrix-containing array is 0, use *Loc0*. However, in a procedure using dynamic matrices *Loc0* must be called.
- Procedures *LOC* and *LOC0* are called when a user program or procedure is handling dynamic matrices of a different and/or varying number of rows. The reader can see that the number of rows of a matrix becomes an input parameter. The difference between *LOC* and *LOC0* is the same as the difference between *Loc* and *Loc0*.

These four procedures can be used to simulate dynamic matrices. They can also be used with arrays on either side of the equal sign in an equation such as:

```
X[Loc(I,J)] := 1.0/Y[Loc(1,J)]
```

The aesthetics of the dynamic matrices are still close to the usual convention. Readability is good. Converting code from other languages will not yield Modula-2 expressions that look very different from the original ones.

The MatMan module offers other matrix management features:

- Swapping columns and rows
- Inserting columns and rows

- Deleting columns and rows
- Transposing matrices
- Re-sizing matrices.

Resizing matrices solves the problem of being locked in with a fixed number of rows. The matrix can expand, assuming that the upper dimension limit is enough, or contract, deleting rows.

If you're interested in MatMan and would like to explore it further, I've uploaded the source code to an automatic test program called MATMAN.TST to the *COMPUTER LANGUAGE* BBS and CompuServe account. This program creates a matrix and manipulates it. After each process, the updated matrix is displayed.

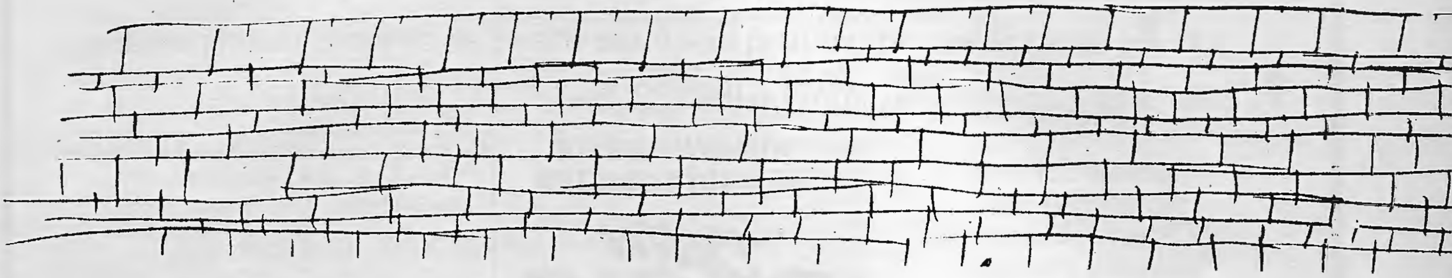
An alternate method to simulate dynamic matrices is suggested by R.S. Wiener.[1] The *ALLOCATE* and *DEALLOCATE* procedures, imported from the module STORAGE, are used to create and remove dynamic matrices. The matrix elements are accessed by pointers. Wiener uses two procedures to access and assign matrix elements values (Listing 2).

The type *Matrix* used in Listing 2 is defined as *ADDRESS*. The speed of these two procedures was reported by Wiener to be on average seven times slower than accessing static matrices.

References

1. Wiener, R. S. "Dynamic Multidimensional Arrays in Modula-2." *Journal of Pascal and Ada* Nov./Dec. (1983).

Namir Shammas is president of his own company, Pyramid Software. He holds a B.S. and M.S. in chemical engineering and enjoys technical writing and programming.



```
IMPLEMENTATION MODULE MatMan;

(*-------------------------------------------------------------------*)
(*               Modula-2 Matrix Management Module                    *)
(*                                                                   *)
(*  Copyright (c) 1984 by  Namir Clement Shammas                     *)
(*  Version 1.0,  August 31, 1984                                    *)
(*-------------------------------------------------------------------*)

PROCEDURE Loc( Row, Col : CARDINAL): CARDINAL;
(* Procedure to locate the (i,j) matrix element, with a default *)
(* HIROW number of rows.                                         *)

BEGIN
    RETURN ( (Col - 1) * HIROW + Row)
END Loc;

(*-------------------------------------------------------------------*)

PROCEDURE Loc0( Row, Col : CARDINAL): CARDINAL;
(* Procedure to locate the (i,j) matrix element, with a default *)
(* HIROW number of rows. Used in subroutines with open arrays.  *)
```

Listing 1 *(Continued on a following page).*

```
PROCEDURE access(a: Matrix; row, col; INTEGER): REAL;
(* Returns the real number in location row,col for matrix a *)

PROCEDURE assign(a: Matrix; row, col : INTEGER; x : REAL);
(* Assigns the real number x to location row,col in matrix a *)
```

Listing 2.

WHY WOULD ANY SANE PERSON SPEND $199 FOR A BetterBASIC SYSTEM WHEN DOS's IS FREE?

HERE ARE 10 REASONS: TEST YOUR SANITY

1. Full support for 640K memory **2.** Structured language with BASIC syntax **3.** Separately compiled program modules **4.** Speed: FAST **5.** Extensibility (Make your own BASIC.) **6.** User-defined procedures and functions **7.** Built-in windows support **8.** Interactive programming language based on an incremental compiler **9.** 8087 math support **10.** Runs on IBM PC, IBM PC/XT and compatibles

Summit Software
Technology, Inc.™
P.O. Box 99, Babson Park
Wellesley, MA 02157

1-800-225-5800

NOW AVAILABLE FOR THE TANDY 2000 & 1200

Better BASIC™

Sane Programmers Order BetterBASIC Now

Price: $199 8087 Math Module: $99
Runtime System: $250 Sample Disk: $10

BetterBASIC is a trademark of Summit Software Technology, Inc. IBM PC, IBM PC/XT and PC/DOS are trademarks of International Business Machines Corp. MS-DOS is a trademark of Microsoft Corp.

MasterCard, VISA, P.O. Checks, Money Orders, and C.O.D. accepted.

CIRCLE 88 ON READER SERVICE CARD

8087 AND 80287 TECHNICAL TOOLS

87FFT™ performs Forward and Inverse FFTs on real and complex arrays which occupy up to 512K bytes of RAM. Also does convolutions, auto correlations, hamming, complex vector multiplication, and complex to radial conversions. Callable from MS Fortran or 87BASIC/INLINE..... **$150**

87FFT-2™ performs two-dimensional FFTs. Ideal for image processing. Requires 87FFT...**$75**

MATRIXPAK™ manages a **MEGABYTE!** Written in assembly language, our runtime package accurately manipulates large matrices at very fast speeds. Includes matrix inversion and the solution of simultaneous linear equations. Callable from MS Fortran 3.2, 87MACRO, 87BASIC/INLINE, and RTOS........ each **$150**

DATA ACQUISITION PACKAGE
Interactive, user-oriented language which allows the acquisition and analysis of large data streams.................................. **CALL**

GRAPHICS PACKAGES

Energraphics (stand alone)	295
Grafmatic for MS Fortran or Pascal	125
Plotmatic for Grafmatic	125
Halo for Basic, C or Fortran	each 150

OTHER TOOLS

Alpha Software ESP	595
Borland Sidekick, Toolbox, or Graphics	45
COSMOS Revelation	850
Lattice C	299
PSI MATHPAK	75
smARTWORK	895
SPSS/PC	695
STSC APL★PLUS/PC	475

RTOS - REAL TIME OPERATING SYSTEM
RTOS is a multi-user, multi-tasking real time operating system. It includes a configured version of Intel's iRMX-86, LINK-86, LOC-86, LIB-86, OH-86, and MicroWay's 87DEBUG. Runs on the IBM-PC, XT, PC-AT and COMPAQ.................. 400

INTEL COMPILERS[1]

FORTRAN-86	750
PASCAL-86	750
PL/M-86	500
87C (LATTICE/MICROWAY)	750
ASM-86	200

URS™ - Universal Run Time System[1]
Generates programs with the Intel compilers which run on other operating systems. MS-DOS version is included with RTOS.
Xenix-286 Version........................... 300

SoftScope Symbolic Debugger[1] 500

[1] Requires RTOS or iRMX-86. All Intel compiler names and iRMX-86 TM Intel Corp.

87BASIC/INLINE™ converts the output of the IBM Basic Compiler into optimized 8087 inline code which executes up to seven times faster than 87BASIC. Supports separately compiled inline subroutines which are located in their own segments and can contain up to 64K bytes of code. This allows programs greater than 128K! Requires the IBM Basic Compiler and Macro Assembler. Includes 87BASIC........... **$200**

PC AT and 86-310 DRIVES

30 MEGABYTE WINCHESTER	2000
53 MEGABYTE WINCHESTER	2600
SYQUEST FIVE MEGABYTE	950

HARDWARE AND LANGUAGES

8087-3 5mhz....... $149
Including DIAGNOSTICS and 180-day warranty
For IBM PC and compatibles

8087-2 8mhz...... $275
For Wang, AT&T, DeskPro, NEC, Leading Edge

80287-3 5mhz.... $275
For the IBM PC AT

64K RAM Set.............. $30

256K RAM Set.......... $195

128K RAM Set PC AT. $225

NUMBER SMASHER™...CALL
10mhz 8087 coprocessor board for the IBM PC

FORTRAN and UTILITIES

Microsoft Fortran 3.2	239
IBM Professional Fortran	595
Intel Fortran-86[1]	750
FORLIB+	65
STRINGS and THINGS	65

BASIC and UTILITIES

IBM Basic Compiler	270
87BASIC/INLINE	200
Summit BetterBASIC™	175
Summit 8087 Module	87

MACRO ASSEMBLERS

IBM Assembler with Librarian	155
87MACRO	150

PASCAL

Microsoft Pascal 3.2	209
Borland Turbo	45
Turbo with 8087 Support	85

NO CHARGE FOR CREDIT CARDS
ALL ITEMS IN STOCK
CALL FOR COMPLETE CATALOG

MicroWay
P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

You Can Talk To Us!

CIRCLE 57 ON READER SERVICE CARD

```
BEGIN
    RETURN ( (Col - 1) * HIROW + Row - 1)
END Loc0;

(*-------------------------------------------------------------------*)

PROCEDURE LOC( Row, Col, HiRow : CARDINAL): CARDINAL;
(* Procedure to locate the (i,j) matrix element, with an assigned *)
(* HiRow number of rows.                                          *)

BEGIN
    RETURN ( (Col - 1) * HiRow + Row)
END LOC;

(*-------------------------------------------------------------------*)

PROCEDURE LOC0( Row, Col, HiRow : CARDINAL): CARDINAL;
(* Procedure to locate the (i,j) matrix element, with an assigned *)
(* HiRow number of rows. Used in subroutines with open arrays.    *)

BEGIN
    RETURN ( (Col - 1) * HiRow + Row - 1)
END LOC0;

(*-------------------------------------------------------------------*)

PROCEDURE Swap( VAR A, B : REAL);
(* Procedure to swap two REALs *)

VAR Tempo : REAL;

BEGIN
    Tempo := A;
    A := B;
    B := Tempo;
END Swap;

(*-------------------------------------------------------------------*)

PROCEDURE SwapColumn(VAR X : ARRAY OF REAL; Col1, Col2, HiRow : CARDINAL);
(* Procedure to swap two columns, Col1 and Col2 *)

VAR i, L1, L2 :  CARDINAL;

BEGIN
    L1 := LOC0(1,Col1,HiRow);
    L2 := LOC0(1,Col2,HiRow);

    FOR i := 0 TO HiRow-1 DO
        Swap(X[L1+i],X[L2+i])
    END;
END SwapColumn;

(*-------------------------------------------------------------------*)

PROCEDURE SwapRow(VAR X : ARRAY OF REAL; Row1, Row2, HiRow, HiCol : CARDINAL);
(* Procedure to swap two rows, Row1 and Row2 *)
```

Listing 1 *(Continued on following page).*

```
VAR i, R1, R2 :  CARDINAL;

BEGIN
    R1 := LOCO(Row1,1,HiRow);
    R2 := LOCO(Row2,1,HiRow);

    FOR i := 1 TO HiCol DO
        Swap(X[R1],X[R2]);
        INC(R1,HiRow);
        INC(R2,HiRow)
    END;
END SwapRow;

(*----------------------------------------------------------------------*)

PROCEDURE Transpose(VAR X, Y : ARRAY OF REAL; HiRow, HiCol : CARDINAL);
(* Procedure to transpose matrix  X into matrix Y *)

VAR i, j : CARDINAL;

BEGIN
    FOR i := 1 TO HiRow DO
        FOR j := 1 TO HiCol DO
                    Y[LOCO(j,i,HiCol)] := X[LOCO(i,j,HiRow)]
    END; (* FOR *)
END; (* FOR *)
```

Listing 1 *(Continued on following page).*

FINALLY...

THE C JOURNAL

The C Journal will help YOU use C on YOUR machine —IBM PC™, CP/M™, Macintosh™, or UNIX™-based — micro, mini, or mainframe.

It's the ONE publication for programmers, software managers, and other computer professionals who need to keep aware of developments in the industry's fastest-growing language.

- regular columns for novice through advanced C programmers
- software product and book reviews
- tips on working with major compilers and operating systems
- news from the ANSI standards committee and the industry

For **FREE** sample issue and discount subscription information, write, call, or circle our reader service number. **The C Journal** is a quarterly publication, and costs $28/year (add $9 for overseas airmail).

Subscriptions/Advertising:
Christina Gardner
(201) 989-0570

Editorial:
Rex Jaesche
(703) 860-0091

another independent publication from
InfoPro Systems
3108 Route 10
P.O. Box 849
Denville, NJ 07834

MasterCard
VISA

CIRCLE 5 ON READER SERVICE CARD

Only $95 with FULL SOURCE CODE!

Q/C

"... an incredible learning tool." *Byte*

For only $95, Q/C is a ready-to-use C compiler for CP/M with complete source code. Here's what *BYTE* (May 1984) said: "Q/C ... has a portable library and produces good code quality. If you want to learn compiler construction techniques or modify the standard language, Q/C is the obvious choice."

- Source code for compiler and over 75 library functions.
- Strong support for assembly language and ROMs.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Q/C is standard. Good portability to UNIX.

Q/C has casts, typedef, sizeof, structure initialization, and function typing. It is compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, or bit fields. Call about our new products: Q/C profiler, Z80 code optimizer, and Z80 assembler and virtual linker, all with full source code!

THE CODE WORKS
5266 Hollister, Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q/C, CP/M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

CIRCLE 42 ON READER SERVICE CARD

```
END Transpose;

(*----------------------------------------------------------------------*)

PROCEDURE InsertColumn(VAR X, Y : ARRAY OF REAL; NewCol : CARDINAL;
                       VAR HiRow, HiCol : CARDINAL);
(* Procedure to insert a new array Y at column NewCol in matrix X *)

VAR i, L, MatSize : CARDINAL;

BEGIN
    IF NewCol <= HiCol THEN
        L := LOCO(1,NewCol,HiRow);
        MatSize := HiRow * HiCol; (* Initial matrix size *)
        (* Move element upward in matrix X *)
        FOR i := MatSize - 1 TO L BY -1 DO
            X[i + HiRow] := X[i]
        END;
        (* Insert column Y *)
        FOR i := 0 TO HiRow - 1 DO
            X[L+i] := Y[i]
        END;
        INC(HiCol);
    END; (* IF *)

END InsertColumn;
```

Listing 1 *(Continued on a following page).*

OVERCOME
FORTRAN and PASCAL
LIMITATIONS WITH

NO LIMIT

$89 Visa/MC

A library of 58 Assembler routines transforms MS FORTRAN and PASCAL plus other 8086/87/88 FORTRANs into the flexible, responsive, complete language needed for the microcomputer environment. Featuring:

EXTENSIVE GRAPHICS	(Get, Put, Paint, Color, Dot, Line, Box, Circle, Ellipse, Large Characters)
FULL SCREEN CONTROL	(Windows, Cursor, Read/Write Screen)
STRING MANIPULATION	(Match, Compare, Concatenate/Extract, Pack, Justify, Zero Fill)
KEYBOARD CONTROL	(Read Key During Execution, String Read With Edit)
FILE MANAGEMENT	(Exist?, Rename, Delete)
COMMUNICATIONS	(Set Com Line, Send/Receive, Line/Modem Status)
OTHER FEATURES	(Peek, Poke, Determine Time/Date, Random Numbers, Beep, Clear Screen, OR/AND/XOR/NOT/NEG of Byte/Word, Printer Status)

For routines to meet language specific problems such as **Chaining** or **Command Line Read** call:

M | E | F Environmental Inc. P.O. Box 26537
Austin, Texas 78755 (512) 251-5543

CIRCLE 55 ON READER SERVICE CARD

SMALL C FOR IBM-PC

Small-C Compiler Version
2.1 for PC-DOS/MS-DOS
Source Code included
for Compiler & Library
New 8086 optimizations
Rich I/O & Standard Library

$40

CBUG SOURCE LEVEL DEBUGGER FOR SMALL C

Break, Trace, and Change
variables all on the
source level
Source code included

$40

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

ASM or MASM is required with compiler.
Include disk size (160k/320k), and DOS version with order.
VISA & MasterCard accepted. Include card no. & expiration date.
Washington state residents include 7.9% sales tax.
IBM-PC & PC-DOS are trademarks of International Business Machines
MS-DOS is a trademark of Microsoft Corporation.

CIRCLE 19 ON READER SERVICE CARD

C

SOFTWARE DEVELOPERS!

V - FILE
THE VIRTUAL MEMORY FILE MANAGER

Let V-FILE save precious development time & cost as you create efficient applications with the power of VIRTUAL MEMORY.

DON'T RE-INVENT THE WHEEL

Why spend weeks or months coding and debugging file and memory management systems when you can order V-FILE today. V-FILE is a library that you can link with your code to provide sophisticated virtual file and memory management — allowing you to concentrate on developing your application.

VIRTUAL DATA OBJECTS SUPPORTED!

Data is referenced by using VIRTUAL MEMORY DATA HANDLES. Your code doesn't need to know whether the data is actually on disk or in RAM. Swapping between disk and RAM and updating files on disk is handled automatically and transparently! Complex VIRTUAL DATA STRUCTURES can be created by linking with data handles instead of pointers.

CHECK THESE FEATURES!

- Multiple, independent swap buffers
- Multiple files per swap buffer
- Highly efficient swap algorithm
- Automatic file updating
- Data prefetching supported
- Data may be locked in memory
- Memory buffers may be flushed
- Makes full use of extended memory on IBM PC/AT
- SOURCE CODE AVAILABLE
- NO ROYALTIES REQUIRED

Supports Dos 2.00+ with Lattice & Microsoft C compilers Supports Microsoft windows

$299

V·FILE™

Contact:
MindBank, Inc.
4620 Henry Street
Pittsburgh, PA 15213
412/683-9800

VISA/MASTER CARD ACCEPTED

CIRCLE 63 ON READER SERVICE CARD

YOUR CODE MAY BE WASTING ITS TIME!
THE PROFILER™ CAN HELP . . .

- **Statistical Execution Profiler**
- **Works with any language**
- **Completely configurable**
- **Up to 16 partitions in RAM/ROM**
- **Time critical code optimization**
- **Abnormal code behavior tracking**
- **Graphic presentation of results**
- **Easy to use menu interface**

THE PROFILER is a software package which gives you, the programmer, a powerful tool for locating time consuming functions in your code and allows you to performance tune your program. With **THE PROFILER** you can determine where to optimize your code for maximum benefit, then measure the results of your efforts.

Using **THE PROFILER,** you can answer questions like:
Where is my program spending its time?
Why is my program so slow? What is it doing?
Is my progam I/O bound? CPU bound? Are data buffers large enough?
How much improvement did my changes make?

THE PROFILER is completely software based and consists of a system resident driver and a monitor program. The memory partitions can range from 1 byte to 1 megabyte in size and can be anywhere in the address space.

NO ADDITIONAL HARDWARE IS REQUIRED!

Requires an IBM PC or compatible system with a minimum 64k and one drive.

THE PROFILER is available for $125.00 **(New Low Price)** from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
THE PROFILER is a trademark of DWB Associates.

dwb
Associates
P.O. Box 5777
Beaverton, OR 97006
(503)629-9645

CIRCLE 48 ON READER SERVICE CARD

DSD80

FULL SCREEN SYMBOLIC DEBUGGER

"THE SINGLE BEST DEBUGGER FOR CP/M-80. A TRULY AMAZING PRODUCT."

LEOR ZOLMAN
AUTHOR OF BDS C

- ☐ Complete Upward Compatibility with DDT
- ☐ Simultaneous instruction, register, stack & memory displays
- ☐ Software in-circuit-emulator provides write protected memory, execute only code and stack protection.
- ☐ Fifteen single keystroke commands
- ☐ Thirty day money back guarantee
- ☐ On-line help & 50 page user manual

ONLY $195.
IBM PC VERSION AVAILABLE SOON!

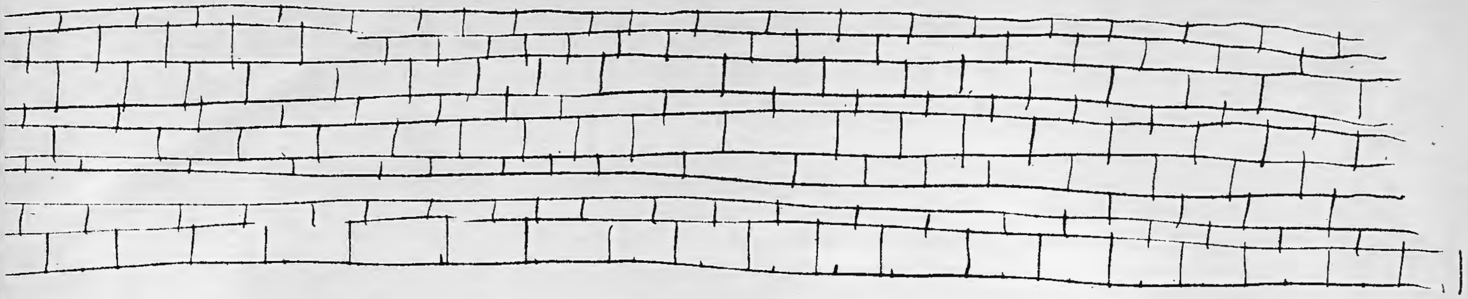
SOFTADVANCES
P.O. BOX 49473 AUSTIN, TEXAS 78765 (512) 478-4763

MasterCard VISA

CIRCLE 77 ON READER SERVICE CARD

```
(*--------------------------------------------------------------------*)

PROCEDURE InsertRow(VAR X, Y : ARRAY OF REAL; NewRow : CARDINAL;
                    VAR HiRow, HiCol : CARDINAL);
(* Procedure to insert a new array Y at row NewRow in matrix X *)

VAR i, j, L1, L2 : CARDINAL;

BEGIN
    IF NewRow <= HiRow THEN
        FOR i := HiCol TO 1 BY -1 DO
            L1 := LOCO(HiRow,i,HiRow);
            L2 := LOCO((HiRow+1),i,(HiRow+1));
            FOR j := 0 TO HiRow - NewRow  DO
                X[L2-j] := X[L1-j]
            END; (* FOR *)
            X[L2 - HiRow + NewRow - 1] := Y[i-1];
            IF NewRow > 1 THEN
                L1 := LOCO((NewRow-1),i,HiRow);
                L2 := LOCO((NewRow-1),i,(HiRow+1));
                FOR j := 0 TO NewRow -2  DO
                    X[L2-j] := X[L1-j]
                END; (* FOR *)
            END; (* IF *)
        END; (* FOR *)
        INC(HiRow)
    END; (* IF *)

END InsertRow;

(*--------------------------------------------------------------------*)

PROCEDURE ResizeMat(VAR X : ARRAY OF REAL;
                    OldHiRow, NewHiRow, HiCol: CARDINAL);
(* Procedure to resize matrix X by reassigning the number of rows *)

VAR i, j, L1, L2, Row : CARDINAL;

BEGIN
    IF NewHiRow < OldHiRow THEN  Row := NewHiRow
                           ELSE  Row := OldHiRow END;
    FOR i := HiCol TO 2 BY -1 DO
        L1 := LOCO(Row,i,OldHiRow);
        L2 := LOCO(Row,i,NewHiRow);
        FOR j := 0 TO Row -1 DO
            X[L2-j] := X[L1-j]
        END; (* FOR *)
    END; (* FOR *)

END ResizeMat;

(*--------------------------------------------------------------------*)

PROCEDURE DeleteColumn(VAR X : ARRAY OF REAL; Col, HiRow : CARDINAL;
                       VAR HiCol : CARDINAL);
(* Procedure to remove column Col from matrix X *)
```

Listing 1 *(Continued on following page).*



```
VAR i, L, MatSize : CARDINAL;

BEGIN
    IF Col <= HiCol THEN
        IF Col < HiCol THEN
            L := LOCO(1,Col,HiRow);
            MatSize := HiRow * HiCol;
            FOR i := L + HiRow TO MatSize - 1 DO
                X[i-HiRow] := X[i]
            END; (* FOR *)
        END; (* IF *)
        DEC(HiCol)
     END; (* IF *)

END DeleteColumn;

(*--------------------------------------------------------------------*)

PROCEDURE DeleteRow(VAR X : ARRAY OF REAL; Row, HiCol : CARDINAL;
                        VAR HiRow : CARDINAL);
(* Procedure to remove row 'Row' from matrix X *)
VAR i, j, L1, L2 : CARDINAL;

BEGIN
    IF Row <= HiRow THEN
        IF Row = HiRow
            THEN ResizeMat(X,HiRow,(HiRow-1),HiCol)
            ELSE
                 FOR i := 1 TO HiCol DO
                     L1 := LOCO(1,i,HiRow);
                     L2 := LOCO(1,i,(HiRow-1));
                     IF Row > 1 THEN
                        FOR j := 0 TO Row-2 DO
                            X[L2+j] := X[L1+j]
                        END; (* FOR *)
                     END; (* IF *)
                     FOR j := Row-1 TO HiRow-1 DO
                         X[L2+j] := X[L2+j+1]
                     END; (* FOR *)
                 END; (* FOR *)
        END; (* IF *)
        DEC(HiRow);
    END; (* IF *)

END DeleteRow;

END MatMan.
```

Listing 1 *(Continued from preceding page).*

WIZARD C

Fast compiles, fast code and great diagnostics make Wizard C unbeatable on MSDOS. Discover the powers of Wizard C:

- ALL UNIX SYSTEM III LANGUAGE FEATURES.
- UP TO A MEGABYTE OF CODE OR DATA.
- SUPPORT FOR 8087 AND 80186.
- FULL LIBRARY SOURCE CODE, OVER 200 FUNCTIONS.
- CROSS-FILE CHECKS OF PARAMETER PASSING.
- USES MSDOS LINK OR PLINK-86.
- CAN CALL OR BE CALLED BY PASCAL ROUTINES.
- IN-LINE ASSEMBLY LANGUAGE.
- 240 PAGE MANUAL WITH INDEX.
- NO LICENSE FEE FOR COMPILED PROGRAMS.

The new standard for C Compilers on MSDOS!

Only $450

WSS

For more information call (617) 641-2379
Wizard Systems Software, Inc.
11 Willow Ct., Arlington, MA 02174
Visa/Mastercard accepted

CIRCLE 86 ON READER SERVICE CARD

PRESENTING THE MEGAMAX C COMPILER

FEATURING:
• IN-LINE ASSEMBLY • ONE PASS COMPILATION • SUPPORT OF DYNAMIC OVERLAYS • FULL ACCESS OF MACINTOSH TOOLBOX ROUTINES • AND MUCH MORE . . .

DEVELOPMENT SYSTEM PACKAGE INCLUDES:
• FULL-SCALE IMPLEMENTATION (K&R) C COMPILER • THE STANDARD C LIBRARY • ROM ROUTINES LIBRARY • LINKER • LIBRARIAN AND DOCUMENTATION . . .

DEALER AND USER GROUP INQUIRES INVITED

NEW FOR THE MACINTOSH

$299.95

FOR MORE INFORMATION OR TO ORDER CALL OR WRITE:

BOX 851521, DEPT. 1
RICHARDSON, TX 75085-1521

Megamax, Inc

(214) 987-4931

MACINTOSH IS A REGISTERED TRADEMARK OF APPLE COMPUTER INC

CIRCLE 13 ON READER SERVICE CARD

MULTI-BASIC

The Compatible BASIC Compiler from Alcor

(Supported Features Chart)

	MBASIC	MBASIC Compiler	CBASIC	CBASIC Compiler	Multi-Basic
IF ELSE/FOR NEXT	●	●	●	●	●
WHILE WEND	●	●	●	●	●
OPEN/CLOSE	●	●	●	●	●
CREATE/DELETE			●	●	●
PRINT/PRINT USING	●	●	●	●	●
LPRINT/LPRINT USING	●	●			●
PRINT @	●	●			●
INPUT/LINE INPUT	●	●	●	●	●
INPUT #/LINE INPUT #	●	●			●
READ #/READ # LINE			●	●	●
LOC/LOF/EOF/ERROR	●				●
FIELD/GET/PUT	●	●			●
RSET/LSET/ERASE	●	●			●
MKD$/MKI$/MKS$	●	●			●
CVI/CVS/CVD/SPC	●	●			●
READ/DATA/RESTORE	●	●	●	●	●
TRON/TROFF	●				●
DEF FN/RANDOM/RND	●	●	●	●	●
DEF USR/SWAP/WAIT	●	●			●
CALL (ASSEMBLY LANG)	●				●
DEFSTR/DBL/SNG/INT	●	●			●
DOUBLE/REAL			●	●	●
INTEGER/STRING			●	●	●
GOTO/GOSUB	●	●	●	●	●
ON ERROR GOTO	●			●	●
RESUME/RESUME NEXT	●				●
ERL/ERR	●			●	●
ON number GOTO/GOSUB	●	●	●	●	●
NAME/RENAME	●	●	●	●	●
PEEK/POKE/INP/OUT	●	●	●	●	●
SYSTEM/SOUND	●				●
SADD/MATCH/UCASE$			●	●	●
VAL/TAB/STR$/VARPTR	●	●	●	●	●
SIN/COS/TAN/ATN	●	●	●	●	●
LOG/EXP/ABS/SQR	●	●	●	●	●
COMMANDS/IF END			●	●	●
INKEY$/INPUT$	●	●			●
TIME$/DATE$/HEX$/OCT$	●	●			●
STRING$/SPACE$	●	●			●
LEFT$/RIGHT$/MID$	●	●	●	●	●
CHR$/ASC/LEN/SGN	●	●	●	●	●
OPTION BASE	●				●
ROW/POS/LPOS	●				●
DIM/MEM/FRE	●	●	●	●	●
MOD/MFRE				●	●
GET/PUT (CHARACTER)			●	●	●
KILL/CLS/CLEAR/INSTR	●	●			●
INT/FIX/CINT	●	●			●
CSNG/CDBL	●	●			●
CHAIN	●		●	●	●
CONSTAT%/CONCHAR%			●	●	●
CONSOLE/LPRINTER			●	●	●
BINARY RANDOM FILES	●	●			●
ASCII RANDOM FILES			●	●	●
255 CHARACTER NAMES				●	●
REDIMENSIONED ARRAYS	●		●	●	●
UNLIMITED STRING SIZE				●	●
MULTI-LINE FUNCTIONS			●	●	●
PROCEDURES/RECURSION					●
FUNCTION TRACING					●
LOCAL VARIABLES			●	●	●
NESTED FUNCTIONS					●
OPTIONAL LINE NUMBERS			●	●	●
DESCRIPTIVE LABELS			●	●	●
SINGLE PRECISION	●	●			●
DOUBLE PRECISION	●	●	●	●	●
LINK TO PASCAL & C					●

For TRS80 models I, II, III, 4, 12, or 2000 using TRSDOS, CP/M, or MSDOS and for IBM PC using PCDOS
Multi-Basic, Pascal, or C $139
Add shipping $6.00 USA, $28 Overseas

ALCOR Systems

13534 Preston Road, Suite 365
Dallas, Texas 75240
(214) 238-8554

Multi-Basic is a trademark of Alcor Systems
TRSDOS, TRS80 is a registered trademark of Tandy Corporation
CP/M, CBASIC are trademarks of Digital Research
MSDOS, MBASIC are trademarks of Microsoft

CIRCLE 1 ON READER SERVICE CARD

Pascal and C Programmers

Your programs can now compile the *FirsTime*™

FirsTime is an intelligent editor that knows the rules of the language being programmed. It checks your statements as you enter them, and if it spots a mistake, it identifies it. *FirsTime* then positions the cursor over the error so you can correct it easily. *FirsTime will identify all syntax errors, undefined variables, and even statements with mismatched variable types.* In fact, any program developed with the *FirsTime* editor will compile on the first try.

Unprecedented
FirsTime has many unique features found in no other editor. These powerful capabilities include a zoom command that allows you to examine the structure of your program, automatic program formatting, and block transforms.

If you wish, you can work even faster by automatically generating program structures with a single key-stroke. This feature is especially useful to those learning a new language, or to those who often switch between different languages.

Other Features: Full screen editing, horizontal scrolling, function key menus, help screens, inserts, deletes, appends, searches, and global replacing.

Programmers enjoy using *FirsTime.* It allows them to concentrate on program logic without having to worry about coding details. Debugging is reduced dramatically, and deadlines are more easily met.

FirsTime for PASCAL	$245
FirsTime for C	$295
Microsoft PASCAL Compiler	$245
Microsoft C Compiler	$395
Demonstration disk	$25

Get an extra **$100 off** the compiler when it is purchased with ***FirsTime.***
(N.J. residents please add 6% sales tax.)

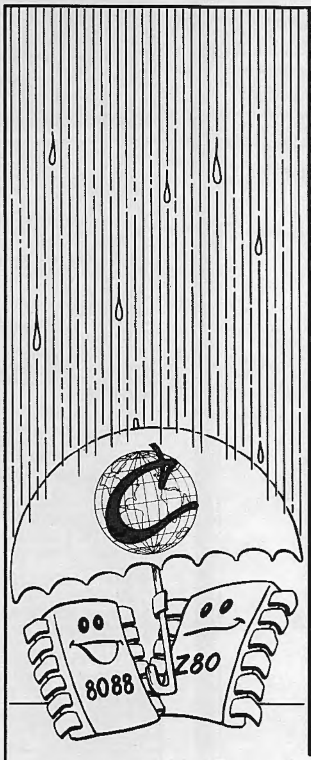
Spruce
Technology Corporation
110 Whispering Pines Drive
Lincroft, N.J. 07738
(201) 741-8188 or (201) 663-0063

Dealer enquiries welcome. Custom versions for computer manufacturers and language developers are available.

FirstTime is a trademark of Spruce Technology Corporation.

MasterCard VISA

CIRCLE 33 ON READER SERVICE CARD



An Optimizing, hassle-free C Compiler for the 8086-8088.

Ecosoft's Eco-C, the performance leader among full C compilers for the Z80, is now available for the 8086-8088 running under MSDOS (2.0 or later). Eco-C has features not found in any other 8086-8088 C compiler.

★ Over 100 library functions. Since they follow UNIX standards, your programs are highly portable in "both" directions ("up" to a UNIX machine or "down" to our Z80 compiler). This means new markets for your software at minimum development cost.
★ A single library. No more "dual" libraries and trying to remember what has to be linked with what. Your programs automatically take advantage of an 8087 if one is present at runtime.
★ A single floating point answer. No more "fuzzy floating point": your programs produce the same answers whether the floating point is done in hardware (8087) or software.
★ Error messages in English. No more cryptic numbers to look up. We tell you where the error occurred, what was found there, and what should have been there.
★ Strict syntax parsing. LINT is going to uncover fewer surprises because our parser looks hard at the details.
★ Efficient code. The optimizer pass of the compiler generates assembler code in Intel mnemonics that rivals that produced by compilers costing twice as much.

The price of the Eco-C compiler is $250.00 (all versions), including the user's manual, and is designed for use with Microsoft's MASM (or compatible) assembler and linker. When ordering, please specify disk format and whether you want the Z80-CP/M or 8088-MSDOS version of Eco-C.

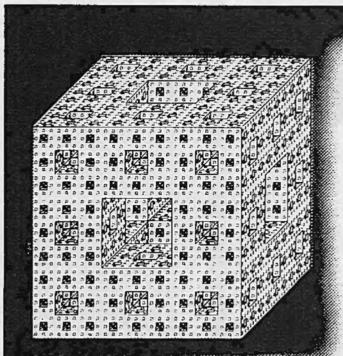
ECOSOFT INC.

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476

MasterCard VISA AMERICAN EXPRESS

Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).

CIRCLE 17 ON READER SERVICE CARD



WALTZ LISP^(TM)

The one and only ***adult*** *Lisp system for CP/M users.*

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with *Franz* (the Lisp running under *Unix*), and is similar to *MacLisp.* Waltz is perfect for Artificial Intelligence programming. It is also most suitable for general applications.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix • True dynamic character strings. Full string operations including fast matching/extraction. • Flexibly implemented random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Virtual function definitions. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Assembly language interface. • Over 250 functions in total. • The best documentation ever produced for a micro Lisp (300+ full size pages, hundreds of illustrative examples).

Waltz Lisp requires CP/M-86 or CP/M 2.2, Z80 and 48K RAM (more recommended). All common 5" and 8" disk formats available.

ProCode (TM)
INTERNATIONAL
15930 SW Colony Pl.
Portland, OR 97224

Unix® Bell Laboratories.
CP/M® Digital Research Corp.

Version 4.4
(Now includes Tiny Prolog written in Waltz Lisp.)

$169*

*Manual only: $30 (refundable with order). All foreign orders: add $5 for surface mail, $20 for airmail. COD add $3. Apple CP/M and hard sector formats add $15.

Call free **1-800-LIP-4000** Dept. #13
In Oregon and outside USA call 1-503-684-3000

CIRCLE 60 ON READER SERVICE CARD

PUBLIC DOMAIN SOFTWARE REVIEW

Disk and library utilities

By Tim Parker

This month we look at a couple of the most frequently used public-domain program utilities in more detail—disk and library utilities.

Disk utilities are immensely useful whenever disk problems occur. Library utilities allow efficient use of disk storage and provide operator convenience. Versions of the programs discussed are available for almost every operating system, as they are among the most useful tools in a programmer's library. Certainly excellent MS-DOS and CP/M versions of these two specific programs are readily available.

A series of library-utility programs have been in common use by programmers for a few years now. These programs are typified by Library Utility (LU) and subsequent versions of it in both CP/M and MS-DOS and its support programs such as Richard Conn's LDIRZ and LRUNZ (part of ZCPR2) for CP/M-80.

Program libraries are essentially the same as programming language libraries: a series of programs (routines, procedures, macros, functions) are all grouped into one large file that contains a directory to the contents and the files themselves. They can be accessed individually, of course, but all reside under the one heading.

Library programs allow similar programs to be grouped together (like a directory tree or dedicated user area). The advantages of doing this are twofold. First, as directories in some operating systems are limited to the number of entries that can be contained per disk (for example, 64 with CP/M-80), grouping files under one library puts the entire collection in the directory as one file, thus increasing the number of files available on a single disk (or hard disk).

This also has a further consequence when telecommunications are involved. Instead of transferring several files separately, one library file can be sent containing them all. When protocol communications are being used, this results in much less operator involvement and an increase in speed. Also, it is much more convenient. This is most noticeable when batch-file transfers are not supported by the communications software.

The second advantage relates to disk space. As operating systems like CP/M and MS-DOS allocate file storage in multiple sectors on a disk (usually 1K or 2K blocks), there can be a lot of wasted space at the end of a file's area. This is obvious if one considers a 2K allocation as standard and realizes that the average file is going to use only half of that space in its last storage sector. (Some files use more, some use less, but statistically they average out to 1K in the last sector.) If there are 20 files in the directory, that leaves unused 20K. As the number of files increases, so does the wasted space.

By grouping the files into a library with only one directory entry, only one sector remains unfilled because the library utilities do not allocate space in the same way as the operating system. Instead, the files are packed into the library using only the sectors they require. The one sector allocated at the end of the entire library by the operating system will average 1K unused, reducing the wasted space from 20K to 1K for a 20-file system.

This may not be important when a hard disk is available, but for small-capacity floppies the difference is an appreciable percentage. Even if the space is not a problem, the convenience of having similarly grouped files together may be important.

One final advantage often overlooked in library utilities is the potential for archival storage. When a diskette library grows to several hundred disks, backups can be not only clumsy but expensive. Library files occupy a smaller amount of space, so a 100-floppy library could be backed up in approximately 50 to 60 floppies.

However, library utilities are not all glory; some problems occur using them. Some program versions do not allow a library entry to be run when inside a library. Instead, it must be extracted from the library into its own file and then executed (if a .COM or .EXE file). It can usually be read when inside a library, though. A number of utilities, such as LRUNZ, have been written allowing these programs to be run from inside the library.

Additionally, some versions require the number of directory entries to be specified when a library is created. This can usually be altered when the library is reorganized, but care should be taken when using this feature. Each version of LU has slightly different features, but they all share the same fundamental command set and perform the same major functions.

Libraries can be used in two modes: interactively and with parameter passing. Basic library functions can be performed, such as adding or deleting library entries, extracting files, listing the library contents, and reorganizational processes. For most functions, wildcards and ambiguous file names are accepted. Further, input from sequential files or source programs (such as BDS C) can be utilized.

One nice feature prevents operator frustration when deleting files; the actual disk directory is not updated until the session has ended. If you have just deleted a very useful program, it is not zapped from the library file itself unless the program is instructed. A warm boot immediately after the erroneous deletion will preserve the library as it was before the deletion.

Combining a number of smaller .COM files into one library and then using a library run program like LRUNZ allows a large number of small utility programs to be maintained and run with little wasted disk space. Typical candidates for this approach are key-definition programs, RAM-disk setups, redirection routines, and similar material.

Now we'll look at disk utilities. As long as there have been programmers, there have been media problems. With the development of the modern high-capacity floppy disk and their frequent access by microcomputer, the message "BDOS ERROR ON B: BAD SECTOR" and its MS-DOS equivalent have become common sights. Trying to get around bad disk problems has been a primary interest for many people.

Despite the rule to back things up frequently, most people inevitably find a bad sector cropping up right in the middle of their favorite file. While many just shrug it off and start again, the ability to recover the lost information is available and being used more frequently.

Bad sectors on disks can be combatted in a couple of ways. The first is to recover the disk's usefulness by placing all the bad sectors into a file that prevents the BDOS from using them.

Programs such as FINDBAD perform this function rapidly and efficiently. The program scans the disk for bad sectors and places them all in a garbage file. Unfortunately, the information in that file is then lost. Also, disk copy programs cannot be used with that disk without more error messages and file transfers must be employed.

Some programs are available that scan a disk much in the same way FINDBAD does and attempt to recover the information in that file. Some move the sector, if possible, to another area on the disk and then place the bad sector in a file of its own. If the problem in the sector is minor, this works well.

Unfortunately, for a truly trashed sector, the problem persists. However, the file is usable except for the bad sectors. If the file is ASCII, then garbage will appear on the screen, but it is usable by word processors. If the file is binary, things are not so good.

Undoubtedly the worst error that can occur to a programmer is a mashed directory track. Then most programs just give up, and the disk is relegated to the garbage can. A second option is available, and it works very well.

This involves accessing the disk sector by sector in some specified manner, ignoring the directory tracks (or if they are still intact, reading them if desired). The material in those sectors can then be moved into a new file and saved on a good disk. This presents the possibility of recovering an entire file, regardless of the condition of the rest of the disk. If a sector in that file is bad though, it can be replaced (by ASCII or binary) at the operator's convenience.

The cost of this power is time. To recover files from a disk requires time to scan the contents of the sectors and group them where they belong. If no directory information is available, this can be quite appreciable! Also, the programs are not the simplest to use. Practice is required, and the manuals have to be read carefully.

Commercial programs that recover files abound. Luckily for the microcomputer user, the best program of all is in the public domain. This is available in a variety of reincarnations, under the names DU, DUU, DU2 and DU3 (and probably others that don't carry the DU designation).

Essentially what the programs all do is read sectors on a disk in any order required in several formats. The contents can be saved into a queue and that queue later written back to a disk as a complete file. Sector, block or group moves are all possible and changing the contents of a particular disk area is relatively easy (assuming you know a little about ASCII codes).

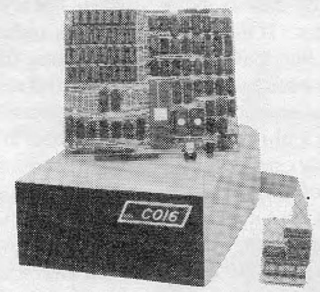
Several functions are available that provide an overview of the disk to be studied, including finding a particular file in the directory if possible and printing its directory-file contents. This shows the sectors allocated to the file, allowing the file to be manipulated without scanning the disk's contents. This usually requires a bit of knowledge about the way the operating system allocates space (for example, understanding the file-control block concepts in CP/M).

More useful than the file find is the map display, which gives a list of the group allocations for files. When invoked, a list of groups on the disk (in ascending numerical order) is displayed, with the associated file name, extent, and user area (if applicable).

Current system statistics can be recalled, including the current disk drive logged in, the number of tracks on the disk, the size of the group in blocks, number of sectors per track, number of groups per disk, number of directory entries, and number of system tracks. Additionally, the queue information that allows saving sectors to new files is displayed.

A POWERFUL 68000 DEVELOPMENT ENVIRONMENT FOR YOUR Z80 SYSTEM

CO1668 ATTACHED RESOURCE PROCESSOR



- **68000 Assembler**
- **C Compiler**
- **Forth**
- **Fortran 77**
- **Pascal**
- **BASIC-PLUS**
- **CBASIC**
- **APL. 68000**

6 MHZ 68000 CP/M-68K 768K RAM
4 x 16081 MATH CO-PROCESSORS CPM80 RAM DISK

Develop exciting 68000 applications on your current Z80 based CPM system using powerful mini-frame like 32 bit programming languages. And then, execute them at speeds that will shame many $100K plus *mini*computer systems.

The **CO1668 ATTACHED RESOURCE PROCESSOR** offers a Z80 CPM system owner a very low cost and logical approach to 68000 development. You have already spent a small fortune on 8 bit diskette drives, terminals, printers, cards cages, power supplies, software, etc. The CO1668 will allow you to enjoy the vastly more powerful 68000 processing environment, while preserving that investment.

CO1668 ATTACHED RESOURCE PROCESSOR SPECIAL FEATURES:

- 68000 running at 6 Mhz
- 256K to 768K RAM (user partitioned between CPU and RAM Disk usage)
- Up to four 16081 math co-processors
- Real time clock, 8 level interrupt controller & proprietory I/O bus
- Available in tabletop cabinet
- Delivered w/ sources , logics, & monolithic program development software
- Easily installed on *ANY* Z80 CPM system
- CP/M-68K and DRI's *new* UNIX V7 compatible C complier (w/ floating point math) - standard feature
- *Can be used as 768K CPM80 RAM Disk*
- Optional Memory parity
- No programming or hardware design required for installation
- Optional 12 month warrantee

- PRICES START AS LOW AS $899.00 for a CO1668 with 256K RAM, CPM68K, C Complier, Sources, Prints, 200 page User Manual, Z80 Interface, and 68000 System Development Software.

For further information about this revolutionary product or our Intel 8086 Co-Processor, please send $1 [no checks please] or call:

HSC INC.

Hallock Systems Company, Inc.
262 East Main Street
Frankfort, New York 13340
(315) 895-7426

RESELLER AND OEM INQUIRIES INVITED.

CIRCLE 31 ON READER SERVICE CARD

Positioning on the disk is handled by a set of movement commands. These allow positioning by group, sector or track or moving to the next or previous physical sector. When moving through a disk, the display shows the current group, track, sector and physical sector.

The contents of a file can be viewed in several formats. ASCII dump is straightforward, but both hexadecimal and ASCII can be combined or used independently for any specified length of blocks (or portions of a block).

When positioned at a disk location, the information can be stored in a memory queue and subsequently examined or written to disk. This allows recovery of files, as shall be explained shortly. The information at the current pointer can be saved to the queue (or removed from it) by block or group. It can be placed either at the head or the tail of the queue. Sections can be removed from the queue and rewritten back to disk just as easily. The queue can be blanked at any time. Queue statistics show the amount of space available, the current size of the queue, and the hex addresses of the queue head and tail. The number of blocks currently in the queue is also reported.

Contents of disks can be changed by specifying the new values to be inserted as hexadecimal numbers, much as with debuggers. A search by string can be performed on specified sectors.

Macro commands are supported in most versions of DU, allowing repeatedly-used command sequences to be summarized into a single command. These macros are made up of the standard commands listed in a file. Macros can be defined at any time during a session.

Recovery of damaged disks follows a fairly straightforward routine. If the directory track is not damaged and the file's location can be determined from that, then the starting address of the file can be reached. Then, as each block is read and verified for its content by the operator, it can be appended to the queue.

If a bad block is encountered in the file, two options are possible. The data can be physically changed at the time using the editing commands, but it is usually easier to simply ignore that section and continue loading the rest of the file to the queue. Then the file is saved to the disk and an editor evoked to replace the damaged section.

When directory information is not available, the disk must be searched for the required file. This is done in a few ways. A string search can be performed if a section of the contents is known, otherwise the disk has to be sampled randomly until a recognizable section is encountered. The starting point of the file can usually then be found by backing up. Then the queue process continues as described earlier.

Recovering files in this method takes time, an average of about 10 to 20 min per file if each sector's contents is examined for inclusion. Further, if the file contains binary, then the task is considerably different from a straight ASCII text file. When several days have been spent writing the thing, the prospect of an hour's work compared to rewriting is usually enough to get the utility started!

Excellent documentation for both the disk and library utilities are available, with lots of samples. However, the disk utility requires practice, and a user is well-advised to use a practice disk before attempting to recover files from a bad disk.

In the coming months we will be taking a look at more standard utilities for both MS-DOS and CP/M and sampling the latest releases from the major public domain distribution groups.

The programs mentioned in this column should be available on both CompuServe and the *COMPUTER LANGUAGE* BBS. Additionally, for those who do not have a modem or do not wish to download the programs, a disk-distribution scheme is being established for much of the material mentioned in this column. Stay tuned for details.

Now with 8087* Floating Point Support

The C Interpreter:
Instant-C™

Programming in C has never been Faster.
Learning C will never be Easier.

Instant-C™ is an optimizing interpreter for the C language that can make programming in C three or more times faster than using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use and learn as Basic. Yet ***Instant-C***™ is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

Instant Editing. The full-screen editor is built into ***Instant-C***™ for immediate use. You don't wait for a separate editor program to start up.

Instant Error Correction. You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only one at a time.

Instant Execution. ***Instant-C***™ uses no assembler or loader. You can execute your program as soon as you finish editing.

Instant Testing. You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

Instant Symbolic Debugging. Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload using special options.

Instant Loading. Directly generates .EXE or .CMD files at your request to create stand-alone versions of your programs.

Instant Floating Point. Uses 8087* co-processor if present.

Instant Compatibility. Follows K & R standards. Comprehensive standard library provided, with source code.

Instant Satisfaction. Get more done, faster, with better results. ***Instant-C***™ is available now, and works under PC-DOS, MS-DOS*, and CP/M-86*.

Find out how ***Instant-C***™ is changing the way that programming is done. ***Instant-C***™ is $500. Call or write for more information.

Rational
Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

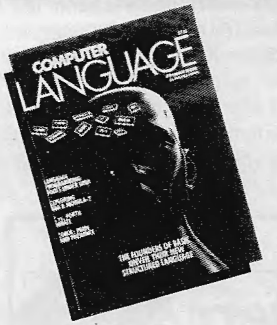
Trademarks: MS-DOS (Microsoft Corp.), 8087 (Intell Corp.), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

CIRCLE 56 ON READER SERVICE CARD

ORDER COMPUTER LANGUAGE BACK ISSUES WHILE THEY LAST!

Complete your collection of COMPUTER LANGUAGE magazines with our selection of back issues. A complete set is sure to become a valuable collector's item in years to come. Here are just a few of the features included in each issue:

PREMIER


COMPUTER LANGUAGE
THE FOUNDERS OF BASIC UNVEIL THEIR NEW STRUCTURED LANGUAGE

The biggest collector's issue

- Basic Becomes a Structured Language—by Kemeny & Kurtz
- Programming in the Unix Environment
- COBOL: Pride and Prejudice
- Exploring Ada and Modula-2

Exotic Language: SNOBOL

Interview: Charles Moore

OCTOBER '84

COMPUTER LANGUAGE

- An Implementation Demonstrating C Portability
- The Evolution of ZCPR–Part I.
- BATCH—A Powerful IBM "Language"

Exotic Language: PILOT

Interview: Donald Knuth

NOVEMBER '84

COMPUTER LANGUAGE
ENHANCING SOURCE CODE CONTROL UNDER UNIX

- Enhancing Source Code Control under UNIX, Part 1
- Natural Language Processing and LISP
- Building Portable Programs

Exotic Language: OCCAM

Interview: Gary Kildall

DECEMBER '84

COMPUTER LANGUAGE
EXPLORATORY PROGRAMMING

- Exploratory Programming
- Fred: A Language within Framework
- Six Pascal Compilers Compared

Exotic Language: OMNI

Interview: Bill Godbout

JANUARY '85

COMPUTER LANGUAGE
MACROS AND PROCEDURES

- Macros and Procedures
- Extensibility in Forth
- The IIIrd Dimension-Programming in dBASE III

Exotic Language: Transaction Application Language

Interview: Sol Libes

FEBRUARY '85

COMPUTER LANGUAGE
SPECIAL C ISSUE

C Language Special Issue

- Twenty-One C Compilers Compared
- The Standardization of C
- C to Assembly Interface

Exotic Language: MUMPS

Interview: P.J. Plauger

Only a limited quantity of magazines is available, so order today. To receive your back issues, just fill out this coupon and mail it back with a check for $4.00 per issue.

Premier ________ copies × $4.00 = $ ________
Oct. '84 ________ copies × $4.00 = $ ________
Nov. '84 ________ copies × $4.00 = $ ________
Dec. '84 ________ copies × $4.00 = $ ________
Jan. '85 ________ copies × $4.00 = $ ________
Feb. '85 ________ copies × $4.00 = $ ________
________ copies × $4.00 = $ ________
________ copies × $4.00 = $ ________
Total $________

NAME ________________
COMPANY ________________
ADDRESS ________________
CITY, STATE, ZIP ________________

Send payment and coupon to:

3

COMPUTER LANGUAGE
Back Issues
131 Townsend St.
San Francisco, CA 94107

THE CODE SWAP SHOP

COMPUTER LANGUAGE
electronic mail

All programs referred to in this reader-inspired, public domain column will be available for downloading when you call the COMPUTER LANGUAGE *Bulletin Board Service at (415) 957-9370—300/1200 baud—or when you dial into CompuServe and invoke our account by typing "GO CLM".*

LZWCOM

Most information processed by computers contains redundancy— repeated characters, words, phrases, etc. Since computers do not have infinite storage or infinite time in which to transfer information, finding ways to remove redundancy from data is attracting much interest.

Kent Williams from Iowa City, Iowa, submits a program this month called LZWCOM, which implements the Lempel/Ziv/Welch data compression algorithm and allows files to be made smaller for transmission and storage.

Users of remote CP/M computers (RCPMs) and other bulletin board facilities may be familiar with the SQUEEZE and UNSQUEEZE programs, the standard way to compress and uncompress information for modem transfer.

These programs use the Huffman encoding algorithm which exploits the fact that certain characters are more likely to appear in a stream of data than others. The input data is translated so that short bit strings represent frequently occurring characters and longer bit strings represent infrequently occurring characters.

Lempel/Zev encoding takes advantage of redundancy on a different level. Instead of viewing data as a stream of characters, it views it as a sequence of strings. A table of strings is built during encoding. The members have this property: if a string

<<string><one_character>>

is in the table, then <string> is in the table as well.

Each element of the string table contains two elements: a predecessor, which is the index of the prefix string, and a follower, which is the suffix character. The string signified by a particular element is composed of the predecessor string plus the suffix string.

Therefore if you start out with a table initialized to one-character strings, you can represent new strings by the index of a string already in the table, plus one character. If a string already in the table is encountered, then the index of the string in the table, rather than the string itself, can be written to output.

This means that it can yield better results than Huffman encoding on data that contains repeated character strings but a nearly even distribution of character frequency, such as object code produced by compilers. Its best-case compression potential is also markedly superior, as one code can theoretically represent a string nearly as long as the table is large, whereas Huffman compression's best case is representing one character by one bit or 1:8.

In general, a table allowing 4,096 strings is used so that a 12-bit code can represent each string. This seems to provide the best trade-off between table size and code length. To use a shorter code would unduly limit the number of possible strings and a longer code would make the table prohibitively large. Also 12-bit codes are relatively easy to pack and unpack with a microcomputer.

The compressor algorithm makes one pass through the source file, progressively biting off larger strings, dynamically building the string table. The uncompressor is able to repeat the process in reverse, proceeding from the fact that it starts from the same table of one character, or atomic strings.

To get this file on the *COMPUTER LANGUAGE* BBS or on CompuServe, look for the file called LZWCOM.DOC in The Code Swap Shop database section.

EDIT

From Minneapolis, Minn., this month comes a new utility program called EDIT, written by Mike Rejsa. This utility is capable of performing many different functions essential for everyday computing, all placed in a single, small, easy-to-use program. Functions include text editing, disk file size and free space reports, file printing, file copying, dumping a file in binary format, and alteration of machine code files. Even if only one of EDIT's functions is used, the program is still smaller than comparable single-function utilities.

A typical application, EDIT used with dBASE II, will enable the user to write and modify command files, check on the disk space left, copy files for backup, list command files on his printer, and (assuming enough knowledge) even examine and repair broken files.

EDIT is unique in that all of these functions are within a single program only 2K in length. Operation is fast and easy with single-letter commands for editing and prompting where required. EDIT will replace PIP, STAT, XDIR, and ED for most users' needs.

One of the reasons for EDIT's small size and fast operation is its method of handling files. The entire file is read into memory at one time and rewritten only after the desired changes are made. For safety, EDIT never changes the original file, always writing a new modified copy. However, the trade-off is that the entire file must fit in the TPA of your computer. For most 64K machines, this means that you can use EDIT on files up to about 50K in size, which is more than adequate for most applications. EDIT won't damage a file too large; it simply won't work with it.

EDIT will work with any CP/M 2.2 or 3.0 computer. If the machine is a CP/M Plus system with bank switching, the amount of disk free space reported will be incorrect, but all other functions work normally. A specific amount of RAM is not required.

To get this program on the BBS or on CompuServe, look for the file EDIT.DOC in The Code Swap Shop area.

Illustration: Anne Doering

DeSmet C

8086/8088 Development Package $109

FULL DEVELOPMENT PACKAGE
- Full **K&R** C Compiler
- Assembler, Linker & Librarian
- Full-Screen Editor
- Execution Profiler
- Complete **STDIO** Library (>120 Func)

Automatic DOS 1.X/2.X SUPPORT

BOTH 8087 AND S/W FLOATING POINT

OVERLAYS

OUTSTANDING PERFORMANCE
- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER $50

- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT $35

- Uses DOS .OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

Check: ☐ Dev. Pkg (109)
☐ Debugger (50)
☐ DOS Link Supt. (35)

SHIP TO: ____________________

____

____________________ ZIP________

CWARE CORPORATION

P.O. BOX C
Sunnyvale, CA 94087
(408) 720-9696

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add $5, elsewhere add $15. Checks must be on US Bank and in US Dollars. Call 9 a.m. – 1 p.m. to CHARGE by VISA/MC/AMEX.

CIRCLE 11 ON READER SERVICE CARD

Thunder Software

- **The THUNDER C Compiler** - Operates under the APPLE Pascal 1.1 operating system. Create fast native 6502 programs to run as stand alone programs or as subroutines to Pascal programs. A major subset of the C defined by K & R. Includes a 24 page users guide, newsletters. Macro preprocessor, runs on APPLE][,][+, //e, //c. Source code for libraries is included. **Only $49.95**
- **ASSYST: The Assembler System** - A complete 6502 editor/assembler and lister for APPLE DOS 3.3. Menu driven, excellent error trapping, 24 p. users guide, demo programs, source code for all programs! Great for beginners. **Only $23.50**
- **THUNDER XREF** - A cross reference utility for APPLE Pascal 1.1. XREF generates cross references for each procedure. Source code and documentation provided. **Only $19.95**

Thunder Software POB 31501 Houston Tx 77231 713-728-5501
Include $3.00 shipping. COD, VISA and MASTERCARD accepted

CIRCLE 65 ON READER SERVICE CARD

MEMO: *C Programmers*

QUIT WORKING SO HARD.

These people have quit working so hard: IBM, Honeywell, Control Data, GE, Lotus, Hospitals, Universities & Government Aerospace.

THE GREENLEAF FUNCTIONS™

THE library of C FUNCTIONS that probably has just what you need . . . **TODAY!**

. . . already has what you're working to re-invent
. . . already has over 200 functions for the IBM PC, XT, AT, and compatibles
. . . already complete . . . already tested . . . on the shelf
. . . already has demo programs and source code
. . . already compatible with all popular compilers
. . . already supports all memory models, DOS 1.1, 2.0, 2.1
. . . already optimized (parts in assembler) for speed and density
. . . already in use by thousands of customers worldwide
. . . already available from stock (your dealer probably has it)
. . . It's called the **GREENLEAF FUNCTIONS.**

Sorry you didn't know this sooner? Just order a copy and then take a break — we did the hard work. Already.

THE GREENLEAF FUNCTIONS GENERAL LIBRARY: Over 200 functions in C and assembler. Strength in DOS, video, string, printer, async, and system interface. All DOS 1 and 2 functions are in assembler for speed. All video capabilities of PC supported. All printer functions. 65 string functions. Extensive time and date. Directory searches. Polled mode async. (If you want interrupt driven, ask us about the **Greenleaf Comm Library.**) Function key support. Diagnostics. Rainbow Color Text series. Much, much more. **The Greenleaf Functions.** Simply the finest C library (and the most extensive). All ready for you. From Greenleaf Software.

. . . **Specify compiler** when ordering. Add $7.00 each for UPS second-day air. MasterCard, VISA, check, or P.O.

◆ Compilers:
CI C86 . . . . $349
Lattice $395
Mark Williams . . . $475

◆ General Libraries $175
(Lattice, Microsoft, Mark Williams, CI C86)
◆ DeSmet C $150
◆ Comm Library $160

GREENLEAF SOFTWARE©

GREENLEAF SOFTWARE, INC.
2101 HICKORY DRIVE ◆ CARROLLTON, TX 75006 ◆ (214) 446-8641

CIRCLE 44 ON READER SERVICE CARD

PRODUCT BINGO

By Doug Millison

Each month Product Bingo features the latest in new software and hardware products of interest to COMPUTER LANGUAGE readers. Product Bingo items are based on information received from the manufacturer and are not meant to be product evaluations, reviews or endorsements. To find out more about a particular product simply circle the appropriate number on the Reader Service card—you'll receive information directly from the manufacturer.

Note to manufacturers: Send new product information to Doug Millison, Product Bingo, COMPUTER LANGUAGE, 131 Townsend St., San Francisco, Calif. 94107.

Windowing on the IBM PC XT/AT

The **Window Machine** from **Amber Systems Inc.** lets IBM PC XT/AT programmers build memory-cheap, but sophisticated, windowing user interfaces into application programs.

Up to 255 windows can be displayed on a physical screen and virtual screens can be as large as 255 columns by 255 lines. ZoomWindows permit a virtual screen to grow or shrink from a single-character cell. A debugging facility permits windowing sequences to be independent of the user application program.

The Window Machine is priced at $59.95.

Amber Systems also announces **Homebase**, a sophisticated desktop organizer featuring communications and enhanced data-base record storage and retrieval features.

Homebase includes a phone-message pad, appointment calendar, calculator, alarm clock, and DOS services. Capable of receiving and storing electronic mail while you're busy with other projects, Homebase also features a cut-and-paste utility you can use to incorporate spreadsheet information into an electronic mail letter.

Available now for the IBM PC/XT and true compatibles, Homebase sells for $49.95, plus shipping and handling.

CIRCLE 101 ON READER SERVICE CARD

PROMAL for the Commodore 64

Commodore 64 programmers can take advantage of **PROMAL, Systems Management Associates**' new high-level structured programming language similar to C and Pascal.

Benchmark results are said to demonstrate that PROMAL is from 70% to 2,000% faster than BASIC, COMAL, Forth, and Pascal.

PROMAL sells for $49.95. The developer's version includes an unlimited run-time distribution license and is priced at $99.95.

CIRCLE 102 ON READER SERVICE CARD

Text editor promises speed, control

Epsilon, a state-of-the-art text editor from **Lugaru Software Ltd.**, is now available for IBM PC, XT/AT, and compatibles running PC-DOS 2.0.

Concurrent process capabilities let you examine and edit files while the compiler runs. Up to 11 windows can be used to display buffers. 125 commands promise total control of all functions.

Epsilon requires 192K bytes of memory and retails for $195.00.

CIRCLE 103 ON READER SERVICE CARD

Convert FORTRAN programs to C

You can convert your old FORTRAN programs and files to C code with **FORTRIX-C** from **Rapitech Systems Inc**.

Priced at $2,750 including manual and telephone support, FORTRIX-C is a development package configured to meet the requirements of various UNIX environments.

Because the generated C code is structured and parallel to the FORTRAN source code, entry-level programmers are said to be able to use FORTRIX-C output to learn C code architecture. Users can move their FORTRAN programs to a UNIX environment at a conversion rate of 600 lines of code per min. FORTRIX-C works with most major computers and can be configured to work with any proprietary system.

CIRCLE 104 ON READER SERVICE CARD

68000 development on IBM PC XT/AT

PC-68K from **Language Resources Inc.** lets IBM PC XT/AT owners develop software for the Motorola 68000 microprocessor.

The base package is priced at $2,995 and includes a Motorola compatible assembler, linker/locator, symbolic debugger, 68000 CPU board with 256K RAM, and documentation. 512K RAM and 1,024K RAM packages are priced at $3,595 and $4,195, respectively.

Pascal and C compilers are available for use with the package at $895 each. A host communication utility ($295) supports two-way transfer of text and binary files between the PC and IBM or VAX mainframes. An emulator communication utility ($495) allows connection of several popular 68000 emulators for debugging software in its final target hardware/software integration.

CIRCLE 105 ON READER SERVICE CARD

SOFTWARE REVIEW

Waltz LISP

Hardware Requirements: 48K CP/M 2.2 or later
Price: $169 including shipping in U.S.
Available from: ProCode International, 16930 SW Colony Place, Portland, Ore. 97224, (800) 547 4000, Dept. 10, (503) 684 3000 in Oregon
Support: By mail for registered owners or leave word at toll-free number.

Waltz LISP, the creation of Dr. Edward Howorka of the Univ. of Virginia, is a remarkable product. With the few exceptions noted, Waltz LISP is a close approximation to Franz LISP implemented in a CP/M environment.

Waltz LISP is powerful, fast, and capable of supporting serious LISP programming. This is partly due to its extremely compact size—only 24K including a 6.5K symbol table—which leaves a useful amount of data and stack space in a CP/M environment. It is currently available for about 65 8-bit machines in 77 different disk formats, so you are likely to be able to obtain Waltz LISP tailored to your machine.

Waltz LISP and Franz LISP have only a few essential differences. One major difference is that Waltz does not currently support real numbers, arrays, or hunks. However, support of reals is promised in a later version.

The elaborate Franz *debug* utility is absent from Waltz to conserve space. But in its place are adequate facilities for examining the stack, tracing functions, setting breakpoints, and extensive error handling capabilities. As a result, *debug* is not likely to be missed. Finally, compilation of Waltz functions is not yet available, although it may be for a later version.

Waltz does include numerous capabilities not found in Franz. These include very powerful matching functions for operations on strings, *peek* and *poke* for the inveterate hacker, the ability to read both text and binary CP/M files, an extended version of *prog* that permits initialization of local variables, a user-defined *undef handler* for intercepting and responding to calls to undefined functions, optional virtual memory for functions, and direct user control of the allocation of memory between data and stack space. In brief, Waltz is an outstanding LISP implementation.

The distribution disk (version 4.40 in my case) contains the files shown in Table 1. In all, about 250 LISP functions are implemented in Waltz and fully documented in the accompanying manual. WLS.COM, the Waltz interpreter, is entirely RAM resident and makes no use of overlays, which further contributes to its speed.

Seven data types are supported including nil, integers, dynamic character strings, file ports, compiled function atoms, cells and lists, and symbols. Unlike some LISP implementations, the maximum integer size supported is from − to + maxint, where maxint = (2**2032) − 1, which is about 611 decimal digits. For practical purposes, this is not a serious limitation, although you can't compute a factorial with an argument larger than 298 without getting an integer overflow error.

The following list provides an overview of Waltz LISP features:

Function definition. Waltz has a uniform function definition format of the form *(def function-name (discipline (parameters) function-body))* where the available disciplines are *lambda*, *nlambda*, *lexpr*, and *macro*. Together these afford all the flexibility in function definition that you are likely to need or want. The Franz function definer *defun*, not included in Waltz, is readily defined if you prefer that syntax.

Initialization. Inevitably you will create additional functions that you want to have included in Waltz whenever it is run. This need is met very elegantly by the provision of an optional initialization file, INIT.WLS, into which you can put such functions. The functions in this initialization file are automatically loaded and evaluated whenever Waltz is invoked.

Waltz functions. The predefined Waltz functions include all of the essential functions found in Franz LISP and a full range of predicate, arithmetic, list, string, and symbol manipulating functions. The complete suite of Franz property list and map-

List of files on the distribution disk

File		Records	Description
#FILES#	.DOC	7	This file
FUNLIST	.DOC	72	List of Waltz LISP's functions
WLS	.COM	192	Waltz LISP interpreter
INIT	.WLS	2	Initialization file (blank)
WLSU	.ASM	134	Assembly language overlay
FDIFF	.	41	File Difference program
FILOPS	.	25	Examples of file operations
FACT	.	9	Example of <syslin> manipulation
UNDEF	.	8	Simple undef-handler definition
SIMEV	.	6	Interpreted <apply> based evaluator
RNDTEST	.	44	Random file access test program
DVIRT0	.	20	Simple virtual function scheme
DVIRT1	.	38	Elaborate virtual function scheme
DVIRTALL	.	5	Define <def> as a virtual definer

Table 1.

ping functions is also present.

Flow of control and iteration. Waltz goes beyond Franz in its provisions for managing the flow of control and iteration. It includes *prog, cond, do, while, until, reptn, catch,* and *throw*. *While* and *until* are functionally the same as *while . . . do* and *repeat . . . until* in Pascal, although their syntax is slightly different. *Reptn* is like *for . . . do* in Pascal and will iterate its body a fixed number of times. It differs from the Pascal *for*, however, in that there is no explicit loop index. None of these latter three functions, which are very convenient, are found in Franz.

Input/output. At the machine level, Waltz handles input and output by the use of four input and four output devices. These devices, individually identified by the integers 0 . . . 3, provide the user with considerable control over the amount and kind of processing that is applied to input and output. You can select the device to be used with *read, print, princ,* or *mapto* by supplying the device number as a parameter in the function call. In addition, the function *drain* is provided to empty the input buffer when that is desired.

The *width* command lets you set the number of columns that will be displayed on the console screen. User-defined file ports are provided to support sequential or random access to disk files and the same port can be used for both reading and writing. In addition, file ports can be defined as text or code ports; the difference is that text ports ignore the high-order bit in each byte, while code ports use the high-order bit, permitting you to read or write binary files from or to the disk.

In addition to the usual *print, princ, map,* and *mapto*, two modes of pretty printing are supported by the functions *prpr* and *pp*, which differ primarily in that *prpr* accepts any list as its argument, while *pp* normally expects a function name; also, *prpr* uses *print* for output, while *pp* uses *princ*.

Limited control over pretty printing indentation is provided by the style function *ps*, which sets or resets a style flag. If the flag is set to nil, indentation is by tab stop. If the flag is set to true, single character indentation is used.

Debugging. Waltz has several debugging tools. These include *ss*, which displays the stack. Depending on the parameters supplied to *ss*, you can selectively display the whole stack, the top of the stack, the contents of any intermediate stack level, the stack contents from the top down to a specified intermediate level, or from an intermediate level down to the bottom of the stack. *trace* is included and permits tracing functions, with trace depth and display mode under user control via parameters to *trace*.

Breakpoints can also be set wherever desired in a function definition to permit halting function execution and examination of the values of variables or parameters, after which function execution can be resumed if desired. Waltz also includes provision for error trapping and, if you wish, the automatic execution of a user-defined series of functions to fix the errors encountered.

Garbage collection. Garbage collection in Waltz is entirely automatic, although it can be invoked at any time by the function *gc*. It is sufficiently fast to be effectively invisible to the user—a consequence, in part, of the fact that it is noncompacting, i.e., nonadjacent free space is not joined together by the garbage collector.

In addition to the foregoing more or less standard LISP functions, Waltz provides numerous additional facilities that make it extremely flexible and convenient to use. Some of the more interesting of these are described as follows:

Disk-related functions. Waltz includes functions to log in a new disk, display the directory of any disk, determine if a named file exists on a given disk, and

For your IBM/PC

mbp COBOL: 4 times faster, and now with SORT & CHAIN.

$750.

mbp COBOL can be summed up in one word: fast.

Because it generates native machine language object code, the mbp COBOL Compiler executes IBM/PC* programs *at least* 4 times faster (see chart).

GIBSON MIX Benchmark Results
Calculated S-Profile
(Representative COBOL statement mix)
Execution time ratio

mbp COBOL	Level II** COBOL	R-M*** COBOL	Microsoft**** COBOL
1.00	4.08	5.98	6.18

128K system with hard disk required. *IBM/PC is an IBM TM; **Level II is a Micro Focus TM; ***A Ryan-McFarland TM; ****A Microsoft TM.

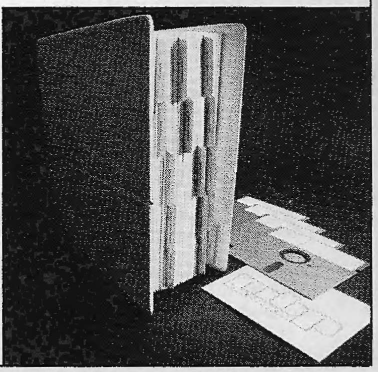
Fast also describes our **new SORT**, which can sort four-thousand 128-byte records in less than 30 seconds. A callable subroutine or stand-alone, 9 SORT control fields can be specified. And our **new CHAIN** is both fast and secure, conveniently transferring control from one program to another, passing 255 parameters. Plus, **new extensions** to ACCEPT & DISPLAY verbs give better, faster interactive programming.

The complete COBOL. An Interactive Symbolic Debug Package included standard; Multi-Keyed ISAM Structure; listing options allow source & object code, map & cross-reference checking; GSA Certification to ANSI '74 Level II; mbp has it all.

It's no surprise companies like Bechtel, Chase, Citicorp, Connecticut Mutual, and Sikorsky choose mbp COBOL; make it your choice, too. mbp is available at Vanpak Software Centers, or direct. For complete information, write **mbp Software & Systems Technology, Inc.**, 7700 Edgewater Drive, Suite 360, Oakland, CA 94621, or phone 415/632-1555 –today.

mbp®

CIRCLE 39 ON READER SERVICE CARD



rename or erase a named file. The directory display and erase function also support ambiguous file names using the usual * and ?.

Memory management. The RAM remaining after Waltz is loaded is partitioned automatically into stack and data space, and the partitioning is adjusted dynamically as Waltz runs. Limited control over the allocation of space is provided by Waltz functions *minst* and *gm*. The function *minst* permits you to specify a minimum size for the stack space. The *gm* function adjusts the memory partitioning to provide the maximum data space that is possible within the constraint of the specified minimum stack size.

Assembly language interface. A limited assembly language interface is provided that permits access to no more than 16 assembly language routines by using the function *sysf* with an integer argument identifying the desired routine and up to three arguments to be passed to the machine language routine. As delivered, *(sysf 5 arg1 arg2)* is already implemented to call BDOS with register C=arg1 and registers DE=arg2, thus giving you direct access from Waltz to all normal BDOS functions.

Incorporation of machine language functions is done directly by using DDT or by modifying WLSU.ASM, which is provided on the distribution disk and as an appendix to the manual. This file contains the modifiable portions of the Waltz machine language kernel and is fully commented and easy to use.

The only problem with this assembly language interface is that WLSU.ASM provides only 96 bytes of patch space where your machine language code can and must be placed. An improved assembly language interface is promised for a later version of Waltz.

Case control. An interesting feature of Waltz is the ability to exercise limited control over how upper and lower case are handled. Three functions implement this feature. The function *cs* sets or resets *csflag*, which if nil causes Waltz—except in strings—to treat upper and lower case as the same. If set to true, the cases are distinguished. In addition, *upperc* will change lower to upper case in the value of its argument, and *lowerc* will do the reverse.

Virtual function definition. The Waltz distribution disk contains three functions—*divert0*, *divert1*, and *divertall*. These functions, fully described in the manual, provide Waltz with the capability of causing new function definitions to be saved to the (real or RAM) disk and automatically recalled from storage when they are invoked from the interpreter.

In view of the limited size of the CP/M transient program area, this capability effectively augments the size of available memory and is especially useful if RAM disk is available.

The documentation accompanying Waltz is not a LISP tutorial but instead is an extensive and impressive technical reference manual consisting of 15 chapters in 265 pages, all in a conventional three-ring binder. The manual is clearly written and contains almost no typographical errors—a real help!

Every Waltz function is described in a standard format, often with accompanying examples of function use. The manual contents are readily accessible either through the detailed 16-page table of contents or by using the excellent 12-page index at the end of the manual.

A detailed listing of error codes with related descriptions is provided as an appendix along with the listing of WLSU.ASM described earlier. In the manual I received, a two-page errata list was also included among the appendices.

For the most part, the descriptions of functions are quite clear and easily understood. Unfortunately, the number of examples of function use seems to be in inverse ratio to the complexity of the functions.

Perhaps this is understandable since complex functions require correspondingly more complex examples to illustrate their features. Nonetheless, in many places examples are sorely missed.

Problems caused by this situation can be mitigated to a large extent by following

GOOD NEWS!

C for the 6809 WAS NEVER BETTER!

INTROL-C/6809,Version1.5

Introl's highly acclaimed 6809 C compilers and cross-compilers are now more powerful than ever!

We've incorporated a totally new 6809 Relocating Assembler, Linker and Loader. Initializer support has been added, leaving only bitfield-type structure members and doubles lacking from a 100% full K&R implementation. The Runtime Library has been expanded and the Library Manager is even more versatile and convenient to use. Best of all, compiled code is just as compact and fast-executing as ever - and even a bit more so! A compatible macro assembler, as well as source for the full Runtime Library, are available as extra-cost options.

Resident compilers are available under **Uniflex, Flex** and **OS9.**

Cross-compilers are available for **PDP-11/UNIX** and **IBM PC/PC DOS** hosts.

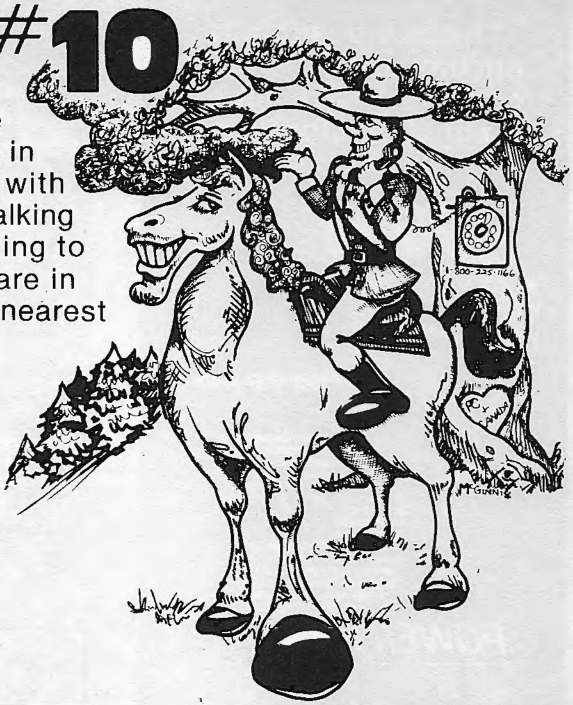
Trademarks:
Introl-C, Introl Corporation
Flex and Uniflex, Technical Systems Consultants
OS9, Microware Systems
PDP-11, Digital Equipment Corp.
UNIX, Bell Laboratories
IBM PC, International Business Machines

For further information, please call or write.

INTROL
CORPORATION
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

CIRCLE 32 ON READER SERVICE CARD

ADVANTAGE #10



Programmer's Connection is proud to announce toll-free telephone service from Canada. We invite programmers in Canada to call 1-800-225-1166 and become acquainted with our service. Calling Programmer's Connection means talking with someone who understands your needs and is willing to take the time to help. And now, no matter where you are in Canada, Programmer's Connection is as close as the nearest phone.

Discover the advantages of buying from Programmer's Connection:

1. We offer the latest version of a product.
2. Most popular products are in stock ready to be shipped.
3. Receive same manufacturer's support as if buying direct.
4. Experienced professional programmers are on staff.
5. Choose from a large selection of the best software products available.
6. Knowledgeable and courteous sales staff.
7. Significant discounts off of retail prices.
8. No extra charge on prepaid orders, including major credit cards.
9. Reasonable charges for shipping and handling.
10. Toll free service from Canada and the Continental U.S.

Programmer's Development Tools:

C LANGUAGE:

	List	Ours
DeSmet C Compiler with Debugger	159	**145**
Lattice C Compiler from Lifeboat	500	**299**
Mac C by Consulair C for MacIntosh	295	**259**
Mark Williams C Compiler w/Source Debugger	500	**449**
Xenix Development System by SCO	1350	**1099**

C.I. **C-86 Compiler and DWB Profiler**
Purchase the C-86 compiler at our regular low price of **$309** and buy the *original* Profiler by DWB Associates for **only $5** (a $125 value).

OTHER LANGUAGES:

	List	Ours
8088 Assembler w/Z-80 Translator-2500 AD	100	**89**
APL*Plus/PC by STSC	595	**499**
Golden Common LISP by Gold Hill	495	**439**
Macro Assembler by Microsoft . *New Release*	150	**119**
Modula-2/86 by Logitech	495	**439**
Professional BASIC by Morgan Computing	95	**89**

PASCAL Starter Kit
An economical way to get started in PASCAL programming. Package includes **Turbo Pascal, Turbo Toolbox,** and **Screen Sculptor** for **only $189!**

C UTILITIES:

	List	Ours
Asynch Comm Library by Greenleaf	160	**124**
C Power Paks from Software Horizons	CALL	**CALL**
C Utility Library by Essential Software	149	**119**
dBC *dBase/C Interface* by Lattice	250	**219**
DOS LINK Support *for DeSmet C*	35	**35**
English-to-C/C-to-English by Catalytix	100	**100**
ES/P for C by Bellesoft	349	**319**
GraphiC by Scientific Endeavors	195	**169**
Greenleaf C Functions Library	175	**129**
Halo *Graphics* by Media Cybernetics	200	**125**
PANEL *Screen Editor* by Roundhill	295	**234**
Run/C Interpreter by Age of Reason	150	**129**
Safe C Standalone Interpreter by Catalytix	CALL	**CALL**
Safe C Dynamic Profiler by Catalytix	150	**150**
Safe C Runtime Analyzer by Catalytix	400	**400**
Windows For C by Creative Solutions	195	**139**

C-SPRITE Source Level Debugger for Lattice C.
List Price $175 Our Price **$159**

c-tree by Faircom
Full featured B-Tree functions for high speed ISAM file management. Comes as *C source code* which can be compiled on almost any system including MacIntosh. *No royalties on generated code.*
List Price $395 Our Price **$359**

OTHER PRODUCTS:

	List	Ours
APL2C by Decision Images *Interfaces APL to C*	150	**139**
Btrieve by SoftCraft	250	**199**
Dr. Halo by Media Cybernetics	95	**79**
FORTRAN Libraries by Alpha Computer Service	CALL	**CALL**
FORTRAN Scientific Subroutine Library	175	**159**
Periscope *Debugger* by Data Base Decisions	295	**269**
Pfix-86 Plus by Phoenix	395	**299**
Plink-86 *Overlay Linker* by Phoenix	395	**299**
Pmate *Macro Text Editor* by Phoenix	225	**159**
Polytron Products . . . . *We Carry a Full Line*	CALL	**CALL**
Profiler by DWB Associates . . *See above offer*	125	**99**
Screen Sculptor by Software Bottling	125	**109**
VSI-Virtual Screen Interface by Amber Systems	60	**54**
XTC *Text Editor* by Wendin	99	**89**
Xtrieve by SoftCraft	195	**169**

CODESMITH-86 Symbolic Debugger by Visual Age
New Version 1.9 now supports In-Line Code Assembly.
Special Price **$119**

Prices are subject to change without notice.

Call for our new Spring Catalog

Account is charged when order is shipped.

MasterCard VISA

1-800-336-1166 In Canada: **1-800-225-1166**

Programmer's Connection
136 Sunnyside Street
Hartville, Ohio 44632
(216) 877-3781 (In Ohio)

"Programmers Serving Programmers"

CIRCLE 27 ON READER SERVICE CARD

"This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
— Dr. Dobb's Journal

C
POWER PACKS™

COMPLETE SOURCES

☐	**PACK 1: Building Blocks I** 250 Functions: DOS, Printer, Video, Asynch	$149
☐	**PACK 2: Database** 100 Functions: B-Trees, Variable Records	$399
☐	**PACK 3: Communications** 135 Functions: Smartmodem™, Xon/Xoff, Modem-7, X-Modem	$149
☐	**PACK 4: Building Blocks II** 100 Functions: Dates, Text Windows, Pull-down Menus Data Compression	$149
☐	**PACK 5: Mathematics I** 35 Functions: Log, Trig, Square Root	$99
☐	**PACK 6: Utilities I** Archive, Diff, Replace, Scan, Wipe (Executable Files only)	$99

Lattice™, Microsoft™, DeSmet™, CI-86™ Compilers on IBM PC/XT/AT™
Small and Large Memory Models.
Credit cards accepted
($7.00 handling/Mass. add 5%)

SOFTWARE HORIZONS INC.

165 Bedford Street
Burlington, Mass. 01803
(617) 273-4711

NOVUM ORGANUM

CIRCLE 25 ON READER SERVICE CARD

Comparative benchmark times (sec)

Benchmark	Waltz	muLISP	IQLISP
Slow append	99	30.6	240
Fibonacci	15.3	8.5	23
Sieve	215	95	*

*Runs out of stack space and aborts.

Table 2.

CP/M-80 C Programmers . . .

Save time

. . . with the BDS C Compiler. Compile, link and execute *faster* than you ever thought possible!

If you're a C language programmer whose patience is wearing thin, who wants to spend your valuable time *programming* instead of twiddling your thumbs waiting for slow compilers, who just wants to work *fast,* then it's time you programmed with the BDS C Compiler.

BDS C is designed for CP/M-80 and provides users with quick, clean software development with emphasis on systems programming.

BDS C features include:

- Ultra-fast compilation, linkage and execution that produce directly executable 8080/Z80 CP/M command files.
- A comprehensive debugger that traces program execution and interactively displays both local and external variables by name and proper type.
- Dynamic overlays that allow for run-time segmentation of programs too large to fit into memory.
- A 120-function library written in both C and assembly language with full source code.

Plus . . .

- A thorough, easy-to-read, 181-page user's manual complete with tutorials, hints, error messages and an easy-to-use index — it's the perfect manual for the beginner *and* the seasoned professional.
- An attractive selection of sample programs, including MODEM-compatible telecommunications, CP/M system utilities, games and more.
- A nationwide BDS C User's Group ($10 membership fee — application included with package) that offers a newsletter, BDS C updates and access to public domain C utilities.

Reviewers everywhere have praised BDS C for its elegant operation and optimal use of CP/M resources. Above all, BDS C has been hailed for it's remarkable *speed.*

BYTE Magazine placed BDS C ahead of all other 8080/Z80 C compilers tested for fastest object-code execution with all available speed-up options in use. In addition, BDS C's speed of compilation was almost *twice* as fast as its closet competitor (benchmark for this test was the Sieve of Eratosthenes).

"I recommend both the language and the implementation by BDS very highly."
Tim Pugh, Jr.
in *Infoworld*

"Performance: *Excellent.*
Documentation: *Excellent.*
East of Use: *Excellent.*"
InfoWorld
Software Report Card

". ... a superior buy . . ."
Van Court Hare
in *Lifelines/The Software Magazine*

Don't waste another minute on a slow language processor. Order your BDS C Compiler today!

Complete Package (two 8" SSSD disks, 181-page manual): **$150**
Free shipping on prepaid orders inside USA.
VISA/MC, COD's, rush orders accepted.
Call for information on other disk formats.

BDS C is designed for use with CP/M-80 operating systems, version 2.2. or higher. It is not currently available for CP/M-86 or MS-DOS.

BDSoftware

BD Software, Inc.
P.O. Box 2368
Cambridge, MA 02238
(617) 576-3828

CIRCLE 4 ON READER SERVICE CARD

the advice that appears early in the Waltz manual and obtaining a copy of *LISPcraft* by Robert Wilensky, published by W.W. Norton & Co. in 1984. This book, based entirely on Franz LISP, could just as well have been put forth as a manual for Waltz LISP since, with obvious exceptions such as reals and arrays and a few minor differences in syntax, most of the examples and exercises in the book can be worked through with Waltz with no or only trivial modification. The book is an invaluable adjunct to Waltz and should be included by ProCode as part of the Waltz package.

The discussion of Waltz performance that follows is presented in the context of my own operating environment, which consists of a Lobo Max-80 running CP/M 2.2. This is a 5 Mhz Z-80 machine that provides a 52K transient program area. In that context, Waltz performance can be described along several dimensions:

Scope. Containing more than 250 predefined functions, Waltz enables you to create any desired LISP functions without resorting to the awkward circumlocutions that would be necessary with a less complete implementation.

Memory utilization. RAM, as used by Waltz, is divided into data and stack space. The data space, in turn, is organized into four-byte cells to accommodate pointers to the *car* and *cdr* of lists. After loading Waltz and manually executing a garbage collection, 3,328 cells are available in data space and 14,573 bytes of stack space are established. The partition between data and stack space is moved up automatically as Waltz runs, with the result that data space grows at the expense of stack space until a stable space allocation is reached.

Since Waltz permits you to establish a minimum stack size and maximize the data space subject to that constraint, simply executing *gm* after loading Waltz maximizes data space and results in 6,144 cells of data space and a stack size of 3,309 bytes. By using *minst* to set minimum stack space before using *gm*, you can make further fine adjustments of the allocation of memory. It should be noted that even when letting Waltz make its space allocation automatically, the resulting data space is large enough to write and execute some quite large LISP functions.

By way of example, the Waltz function editor described later in this review takes up about 6K on disk and occupies only 385 cells in Waltz. In addition, for every 1K bytes of TPA space that you add, say by using CP/M 3.0 rather than 2.2, you can get an additional 256 cells. The compactness of Waltz really pays off and lets you do some significant "LISPing" in a CP/M environment.

Renaming functions. Since Waltz and Franz have some differences in function names (e.g., *add* and *div* instead of *plus* and *quotient*), you may prefer to use the Franz names for those functions. This is easily accomplished. For example, the simple construction *(defplus add)* makes *plus* an alias for *add*. If desired, a series of such expressions can be included in INIT.WLS so that they will be automatically loaded and evaluated when Waltz is invoked.

Undefining functions. If you have had a long session with Waltz it is not unusual to have a number of no longer useful functions cluttering up memory and taking up valuable data space. This space is easily recovered by simply entering *(def function-name nil)* for each function you no longer need. These expressions will un-define the designated functions, and the space they occupied will be recovered by the garbage collector. This is a valuable capability if data space becomes tight.

Garbage collection. As remarked earlier, Waltz garbage collection is extremely fast, taking only about one second. As a result, when garbage collection is automatically invoked it has no material impact on the execution speed of Waltz functions.

Btrieve and Xtrieve.
B-tree file access for all your programming languages. With the query tool your users demand.

Introducing a powerful data base combination for PC application developers. Btrieve™, the most sophisticated file access method for your IBM™ PC. In single user and network versions. And Xtrieve™, a new menu-driven query tool that gives you—and your users—fast access to information.

Btrieve: for professional programmers. Btrieve provides fast, flexible file management for all your application development. All your programming languages—BASIC, Pascal, Cobol, C. With multikey access to records. Automatic file recovery. Unlimited records per file. Duplicate, modifiable, and segmented keys.

With Btrieve, you can develop *better* applications *faster.*

Xtrieve: easy window interface. The ideal complement to Btrieve, Xtrieve is the non-programmer's interface. Xtrieve's full relational capabilities let users define a virtual table of data from multiple files. Then Xtrieve speeds them through query building with a series of easy-to-follow windows.

No command language. Xtrieve is completely menu-driven, so there's no need to memorize command language. Or special syntax. Everything you need is on the screen.

Xtrieve features a full range of restriction criteria. Online help messages. And interfaces to access information from Lotus 1-2-3™ and dBase II® files.

Network capabilities. Network versions of Btrieve and Xtrieve allow data sharing in PCnet™, NetWare™, EtherSeries™, MultiLink™, and OmniNet™ LANs.

For more information or to order, contact:

SC SoftCraft Inc.
P. O. Box 9802 #590 Austin, Texas 78766 (512) 346-8380

Suggested retail prices: Btrieve, $245; Xtrieve, $195. Btrieve/N (network), $595; Xtrieve/N, $395. Dealer inquiries welcome. Btrieve requires PC-DOS or MS™-DOS 1.X or 2.X; Xtrieve, PC-DOS or MS-DOS 2.X.

Btrieve and Xtrieve, IBM, 1-2-3, dBase II, PCnet, NetWare, EtherSeries, MultiLink, OmniNet, and MS are trademarks of SoftCraft Inc., International Business Machines, Lotus Development Corp., Ashton-Tate, Orchid Technology, Novell Data Systems, 3Com Corp., Davong Systems Inc., Corvus Systems, and MicroSoft Inc.

CIRCLE 78 ON READER SERVICE CARD

Benchmarks. LISP benchmarks are hard to come by, but the July 1984 issue of *BYTE*, page 281, contains a review of muLISP and IQLISP for the IBM PC and includes the results of several benchmarks. (It should be noted that muLISP requires 128K bytes of memory and IQLISP requires 256K!)

Table 2 shows a comparison of the execution times for Waltz and these LISPs for three benchmarks: a specially constructed, very slow version of *append*; computation of the first 17 terms of the Fibonacci series; and a recursive version of the Sieve of Eratosthenes executed with an array size of 500. As is evident from the table, Waltz delivers a very creditable performance in comparison with these larger implementations.

Moreover, as you will discover by experience, the Waltz function editor, which is a reasonably big LISP function, executes essentially instantaneously even when editing a large function such as *edit* itself. You will not be disappointed by the speed of Waltz.

Editing. If Waltz has a deficiency, it is the absence of an integrated editor, although one is promised for a later version of the interpreter. In the meantime, since Waltz saves and loads functions in ASCII format, you will want to use a full screen text editor to write the initial version of any large LISP functions. WordStar will work in the non-document mode, as will Nevada Edit, Vedit, Mr. Edit, and numerous others. Nevertheless, once you have written the initial version of a function, you are likely to want to have a function editor that is directly accessible from within Waltz.

EDIT, a simple Waltz function editor, is provided in Listing 1. In use, it is invoked by *(edit 'f-name)*. Note that the name of the function to be edited must be quoted.

The editor operates by constructing the list representation of the function designated by *f-name* and passing it to *ed*, which is a general-purpose list editor. On entry, *ed* displays a level indicator consisting of an integer identifying the level you are currently at, followed by a greater than sign, e.g., 1>. This is followed by the first element of the top-level list, which of course will be *def*. *ed* then skips two lines and displays a ? to prompt for your command.

You can then move forward, backward, down, and up in the list using the cursor control keys. After each command, the screen will display after the level indicator the list element at your current location in the list. If you go down in the list or up, the level indicator will change to display the new current level.

The function being edited can be modified wherever desired, renamed, displayed or printed, or executed by using the commented single key commands shown after *selectq* in the listing. Changes made at the current level can also be cancelled if desired, but this must be done before leaving the level.

If an error results when the function is executed by typing "x", an appropriate error message will be displayed on the console, and you will be returned to your place in the editor ready to continue. If the function being executed with "x" requires two or more arguments, these must be entered as a list, i.e., enclosed in parentheses. When you have made all the changes you desire, get back to the top level and type "q" to quit the editor and save your changes. The edited function is then ready for use.

Note that since *edit* is fully recursive it can be used to edit itself, i.e., type *(edit 'edit)*. This provides a convenient way to modify the editor in any way you choose, for example, by replacing the control characters for my cursor keys by those appropriate to your machine. You may also wish to extract *ed* for use separately as a general-purpose list editor.

In conclusion, the deficiencies of the present version of Waltz have been enumerated. It lacks reals, arrays, and hunks; it can't be compiled; it has an awkward and limited assembly language interface; and it lacks an integral function editor. On the other hand, it is well-documented, fast, compact, and more complete than I would have believed possible in 24K. In brief, it's real LISP in a CP/M environment, offered at an affordable price. I thoroughly recommend it. ■

By R.A. Langevin

C + C UTILITY LIBRARY = PRODUCT

• We have over 200 complete, tested, and, documented functions. All source code and demo programs are included.

• The library was specifically designed for software development on the IBM PC, XT, AT and compatibles. There are no royalties.

• Over 95% of the source code is written in C. Experienced programmers can easily "customize" functions. Novices can learn from the thorough comments.

We already have the functions you are about to write

Concentrate on software development—not writing functions.

THE C UTILITY LIBRARY includes:

• Best Screen Handling Available • Windows • Full Set of Color Graphics Functions • Better String Handling Than Basic • DOS Directory and File Management • Execute Programs, DOS Commands and Batch Files • Complete Keyboard Control • Extensive Time/Date Processing • Polled ASYNC Communications • General DOS/BIOS gate • And More •

• The Library is compatible with: Lattice, Microsoft, Computer Innovations, Mark Williams and DeSmet.

C Compilers: Lattice C—$349, Computer Innovations C86—$329; Mark Williams C—$449.

C UTILITY LIBRARY $149

Order direct or through your dealer. *Specify compiler* when ordering. Add $4.00 shipping for UPS ground, $7.00 for UPS 2-day service. NJ residents add 6% sales tax. Master Card, Visa, check or P.O.

ESI
ESSENTIAL SOFTWARE, INC
P.O. Box 1003 Maplewood, New Jersey 07040 914/762-6605

CIRCLE 18 ON READER SERVICE CARD

```
;EDIT (A Simple Waltz Function Editor)
;By R. A. Langevin, 7621 Fontaine Street, Potomac, MD 20854
;Version 1.4, 7 September 1984

(def edit (lambda (f-name)
   (prog (flist (level 1))
   (def ed (lambda (l)
       (prog (head here tail z)
       cancel
          (setq head nil)
          (setq here (car l))
          (setq tail (cdr l))
       loop
          (msg (-2) (1 level) (-2 ">") (t (-1 here)) (-2) (-2 "?") )
          (errset (selectq (readc 3)

             (^_  (cond   ;right arrow = move right in list
                     [(null tail) (princ "at end of list")]
                     [t (setq head (reverse (cons here (reverse head))))
                        (setq here (car tail))
                        (setq tail (cdr tail))]))

             (^^  (cond   ;left arrow = move left in list
                     [(null head) (princ "at start of list")]
                     [t (setq tail (cons here tail))
                        (setq here (car (last head)))
                        (setq head (reverse (cdr (reverse head))))]))

             (^]  (cond   ;down arrow = move down in list
                     [(not (dtpr here)) (princ "no lower level")]
                     [t (setq level (add1 level))
                        (setq here (ed here))]))

             (^\  (cond   ;up arrow = move up in list
                     [(equal level 1) (princ "at top level now")]
                     [t (setq l (append head (list here) tail))
                        (setq level (sub1 level))
                        (return l)]))

             (i   (cond   ;toggle prpr indentation level
                     [(ps) (ps nil) (princ "wide indentation")]
                     [(not (ps)) (ps t) (princ "narrow indentation")] ))

             (s           ;show list at current level on console
                  (print (append head (list here) tail)) )

             (w           ;prettyprint list at current level on console
                  (prpr (append head (list here) tail)) )

             (h           ;prettyprint list at current level on printer
                  (prpr (append head (list here) tail) 2) )

             (c           ;cancel changes at current level
                  (go cancel) )
```

Listing 1 *(Continued on following page).*

```
            (b           ;go to beginning of list
                 (setq z (append head (list here) tail))
                 (setq head nil) (setq here (car z)) (setq tail (cdr z)) )

            (e           ;go to end of list
                 (setq z (append head (list here) tail))
                 (setq tail nil) (setq here (car (last z)))
                 (setq head (reverse (cdr (reverse z)))) )

            (d   (cond   ;delete here
                    [(null tail) (setq here (car (last head)))
                     (setq head (reverse (cdr (reverse head))))]
                    [t (setq here (car tail)) (setq tail (cdr tail))] ))

            (r           ;replace here
                 (princ "replacement = ")
                 (setq here (progl (read) (drain))) )

            (p           ;insert preceding here
                 (princ "insertion = ")
                 (setq tail (cons here tail))
                 (setq here (progl (read) (drain))) )

            (f           ;insert following here
                 (princ "insertion = ")
                 (setq head (reverse (cons here (reverse head))))
                 (setq here (progl (read) (drain))) )

            (n   (cond   ;assign new function name
                    [(greaterp level 1) (princ "not at top level")]
                    [t (princ "new function name = ")
                       (setq head (list 'def))
                       (setq here (progl (read) (drain)))
                       (setq tail (cddr l))] ))

            (x   (cond   ;execute current version of function
                    [(greaterp level 1) (princ "not at top level")]
                    [t (setq z (append head (list here) tail))
                       (eval z) (princ "argument list = ")
                       (eval (append (list (cadr z)) (read)))
                       (drain)] ))

            (q   (cond   ;quit edit and save changes
                    [(equal level 1) (setq l (append head (list here) tail))
                     (return l)]
                    [t (princ "not at top level")] ))

            (princ "wrong key, try again") ))
       (go loop))))

;begin the body of edit itself

   (setq flist (append (list 'def f-name) (list (getd f-name))))
   (setq flist (ed flist))
   (return (eval flist))]
```

Listing 1 *(Continued from preceding page).*

STRUCTURE FOR BASIC
With PROFESSIONAL PROGRAMMING ENVIRONMENT

BENDORF ASSOCIATES
6006 S. MAIN
P.O. BOX 5910
ROSWELL, NM
88201
505 347-5701
VISA/MASTERCARD

ELIMINATES LINE NUMBERS
ALLOWS MULTI-LINE CONDITIONALS
PCDOS/MSDOS

WORKS WITH BASICA INTERPRETER COMPILER
FULL ERROR LOGGING
PROGRAM LISTER

LABELED PROCEDURES
MACROS
SUB-ROUTINES
LIBRARIES

$49.95

CIRCLE 6 ON READER SERVICE CARD

BYSO™ LISP

has features that will delight both beginners and advanced programmers. A fast, reliable and complete interpreter for the IBM PC and true compatibles. $125 includes 100 pg. ref. manual and application notes that put you months ahead on useful projects *(making a hybrid language with C, accessing system functions and I/O ports, building your own dialect, etc.).*

LEVIEN INSTRUMENT CO.
P.O. Box 31G
McDowell, VA 24458
703-396-3345

IBM PC is a trademark of the IBM Corp.

A general purpose programming language for string and list processing and all forms of non-numerical computation.

SNOBOL4+ —the entire SNOBOL4 language with its superb pattern-matching facilities • Strings over 32,000 bytes in length • Integer and floating point using 8087 or supplied emulator • ASCII, binary, sequential, and random-access I/O • Assembly Language interface • Compile new code during program execution • Create SAVE files • Program and data space up to 300K bytes RAM

Have you tried SNOBOL4+?

MUSIC ANALYSIS • TEXT PROCESSING • SYMBOLIC MATHEMATICS • ARTIFICIAL INTELLIGENCE • COMPILER PROTOTYPING • CRYPTOGRAPHY • LINGUISTICS • LIST PROCESSING • GAMES •

With **ELIZA** & over 100 sample programs and functions

For all 8086/88 PC/MS-DOS or CP/M-86 systems, 128K minimum 5¼" DSDD, specify DOS/CPM format

Send check, VISA, M/C to:
Catspaw, Inc.
P.O. Box 1123 • Salida, CO 81201 • 303/539-3884

$95
plus $3 s/h

CIRCLE 9 ON READER SERVICE CARD

CALL FOR PAPERS on Artificial Intelligence/ Expert Systems

Special issue this July

Send to: Editor
COMPUTER LANGUAGE
131 Townsend St.
San Francisco, Calif. 94107
(415) 957-9353
BBS#: (415) 957-9370
CompuServe acct.: GO CLM

ADVERTISE
in the May issue of
COMPUTER LANGUAGE
Special BASIC issue

Reservation deadline:
March 6

Contact:
Carl Landau
COMPUTER LANGUAGE
131 Townsend St.
San Francisco, Calif. 94107
(415) 957-9353

BDS C v1.5
For *FAST* Development

- Fastest compile to execute cycle of any CP/M 80® Compiler.
- Dynamic debugger.
- 180 page manual.

KS res. add 4% tax.
$120.00
Plus $2.50
Shipping & handling

TERMS: check, c.o.d., charge card.
(316) 431-0018

DMS
Dedicated Micro Systems, Inc.
P.O. Box 481 Chanute, KS 66720

CP/M ® trademark of Digital Research

CIRCLE 21 ON READER SERVICE CARD

C Programmers
B-Trees
For
$75.00 + 2.00 Postage
Source Code Included

The Softfocus B-Trees record indexing library will help you develop sophisticated application programs. With Softfocus B-Trees you get:

- high speed file handling for up to 16.7 million records per file
- customizable BDS or K & R standard C source code
- no royalties on applications programs
- support random and sequential file access
- includes example programs

Softfocus
1277 Pallatine Dr., Oakville, Ont.
Canada L6H 1Z1 (416) 844-2610

CHARGEX
VISA

CIRCLE 79 ON READER SERVICE CARD

YOU CAN C

VIEW v3
The ultimate CP/M disk utility.

- **Edits** any disk sector
- **Displays** allocation bit map
- **Rebuilds** files automatically from selected blocks

If you ever had to recover a crashed disk, you need **VIEW.** Use **VIEW** to patch damaged sectors, un-erase files, examine other disk formats, more.
COMPLETE C SOURCE CODE $59

BD Software's C Compiler
The most popular CP/M C Compiler comes with **symbolic debugger** and special Western Wares' **ISIS-II** function library **$140**

C-PACK
A disk full of useful CP/M utilities written in C, PEPE, DEL, BACKUP, DPATCH, ECHO, CHX, PAUSE, more Source incl **$19**

Other C Products
EZZAP, ICX, MEDIT, more
Send for a catalog today

CP/M®, Digital Research
ISIS-II ®, Intel Corp

Western Wares 303-327-4898
Box C • Norwood, CO 81423

CIRCLE 41 ON READER SERVICE CARD

↓
→ *All* LOMAS DATA PRODUCTS' ←
16-Bit. S-100 Bus: Systems & Boards

* * * *FIREBALL DISCOUNTS* * * *

25% off List Price (UPS extra) for Cashier's Check or Money Order
+ + +
You save *MONEY* = We save *TIME*
+ + +
See → LOMAS ← ad in BYTE or request complete specifications
ϟ
\ *LIGHTNING SPEED* /
80286-CPU & 80287-NDP: *$1458.75*

! ! *THUNDERING PERFORMANCE* ! !
80186 & I/O & 256K-DRAM & 5.25"/8" disk controller & CCP/M-86: *$1196.25*

- - - - - - - - - - - - - - - - - - -

Configured for *SanTec-S700* printer:
TeleVideo-GA970C VDT: *$1121.75*
Spellbinder-5.30 W/P: *$371.25*
- - -
HIERATIC COMPUTER SYSTEMS
BOX 133; MEDFORD, MA 02155
(617) 683-6540

CIRCLE 45 ON READER SERVICE CARD

ADVERTISER INDEX

	PAGE NO.	CIRCLE NO.
Alcor Systems	61	1
Amber Systems	2	2
Awareco	51	3
BD Software	74	4
Bendorf & Associates	79	6
Blaise Computing Inc.	49	8
Borland International	1	10
C Journal	56	5
C Systems	41	16
C Ware	68	11
Catspaw Inc.	79	9
Code Works (The)	56	42
CompuServe	4	7
Computer Helper Industries	10	12
Compuview Products	34	81
DWB Associates	58	48
Data Base Decisions	51	30
Datalight	57	19
David Smith Software Co.	10	36
Dedicated Micro Systems Inc.	79	21
Ecosoft	62	17
Essential Software Inc.	76	18
FairCom	12	29
Gimpel Software	14	50
Greenleaf Software	68	44
HSC Inc.	64	31
Harvard Softworks	20	47
Hieratic Computer Systems	79	45
Information Systems Inc.	44	20
Introl Corp.	72	32
Laboratory Microsystems Inc.	12	35
Lahey Computer Systems Inc.	10	22
Lattice Inc.	48	53
Levien Instrument Co.	79	-
Lomas Data Products	8	54
mbp Software & Systems Technology	71	39
MEF Enviromental	9	15
MEF Enviromental	57	55
Manx Software Systems	6	69
Megamax Inc.	61	13
Micro Methods	35	52
MicroWay	54	57
Miller Microcomputer Service	21	49
Mindbank, Inc.	58	63
Northwest Computer Algorithms	35	46
Oasys	35	24
Phoenix Compter Products Corp.	25	82
Poor Person Software	35	26
ProCode	62	60
Programmer's Connection	73	27
Programmer's Shop (The)	36	94
RR Software	Cover II	58
Rational Systems Inc.	65	56
Realia Inc.	50	76
Seven Seas	24	28
Shaw American Technologies	22	34
Simpliway Products Co.	24	70
Soft Advances	58	77
Soft Craft Inc.	75	78
Softfocus	79	79
Software Horizons	74	25
Software Toolworks	80	37
Solution Systems	17	38
Solution Systems	17	93
Solution Systems	17	97
Solution Systems	17	99
Spruce Technology Corp.	62	33
Stride Micro	Cover III	72
Summit Software	54	88
Systems Guild	23	89
Thunder Software	68	65
UniPress	44	40
Viasyn Corp./CompuPro	Cover IV	91
Western Wares	79	41
Whitesmiths Ltd.	18	43
Wizard Systems	61	86
Workman & Associates	22	68

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

"C/80 . . . the best software buy in America!"
—MICROSYSTEMS

Other technically respected publications like *Byte* and *Dr. Dobb's* have similar praise for **The Software Toolworks' $49.95** full featured **'C' compiler** for CP/M® and HDOS with:

- I/O redirection
- command line expansion
- execution trace and profile
- initializers
- Macro-80 compatability
- ROMable code
- and much more!

"We bought and evaluated over $1500 worth of 'C' compilers . . . C/80 is the one we use."

—Dr. Bruce E. Wampler
Aspen Software
author of *"Grammatik"*

In reviews published worldwide the amazing **$49.95 C/80** from **The Software Toolworks** has consistently scored at or near the top — even when compared with compilers costing ten times as much!

The optional **C/80 MATHPAK** adds 32-bit floats and longs to the C/80 3.0 compiler. Includes I/O and transcendental function library all for only **$29.95!**

C/80 is only one of 41 great programs each **under sixty bucks.** Includes: LISP, Ratfor, assemblers and over 30 other CP/M® and MSDOS programs.

For your **free** catalog contact:
The Software Toolworks
15233 Ventura Blvd., Suite 1118,
Sherman Oaks, CA 91403 or call 818/986-4885 today!

CP/M is a registered trademark of Digital Research.

CIRCLE 37 ON READER SERVICE CARD

"Despite the recent press notices, multiuser microcomputers aren't anything new!"

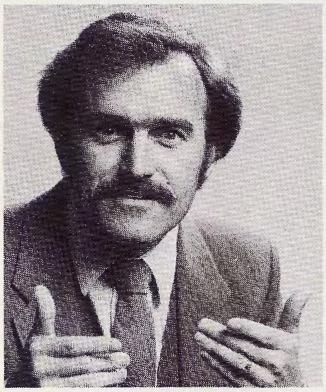
This is the first in a series of discussions with Rod Coleman, President of Stride Micro (formerly Sage Computer) on the 68000 multiuser market and its current environment.

Q: Why do you say that?

RC: "The technology to build a high performance multiuser system has been around for five years. And while some of the leaders in this industry have been pretending that micro multiuser didn't exist, we've been shipping complete systems for nearly three years. The benefits of multiuser are undeniable; it is more cost effective, and offers greater flexibility and utility. But until just recently, the marketing pressure to be compatible instead of being better, has blinded the industry."

Q: What do you mean?

RC: "Well, for example, the Motorola 68000 processor introduced 16/32-bit technology to the personal computer world a long time ago. It was fully capable of meeting high performance and multiuser design requirements in 1980. Instead of this trend taking off, most energy was spent promoting 8088/8086 products that were clearly inferior from a technical point of view. This phenomenon leads me to believe that they will soon rewrite the old proverb: 'Build a better mousetrap and the world will beat a path to your door,' but only if they can find the way through the marketing fog."



"A surprising feature is compatibility. Everybody talks about it, but nobody does anything about it."

Q: Are things changing now?

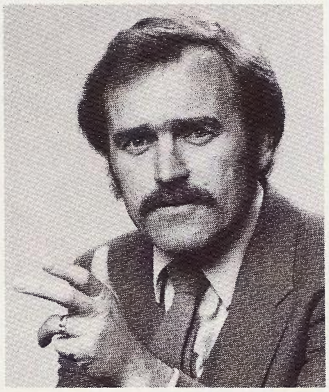
RC: "Yes and no. With the business world starting to take more and more interest in microcomputer solutions, the advantages of a solid multiuser system couldn't be kept hidden forever; companies like ours and a few others were beginning to make a dent. Instead of taking a fresh approach, some of the newest multiuser offerings will probably only give the technology an undeserved black eye! Multiuser is far more than the ability to plug in more terminals. It involves things like machine compatibility, fast processors, adequate memory, large storage capacities, backup features, networking, and operating system flexibility."

Q: Is this what makes the new Stride 400 Series different?

RC: "Exactly. That sounds self-serving, but it's true. Today a number of companies are introducing their first multiuser system. We've been building and shipping multiuser machines for almost three years. We know the pitfalls, we've fallen into some of them. But we have learned from our mistakes."

Q: Give me some examples.

RC: A hard disk is almost mandatory for any large multiuser installation. Yet, backing up a hard disk can be a nightmare if you only have floppies to work with. That's why we've added a tape backup option to all the larger Stride 400 Series machines. It's irresponsible for a manufacturer to market a multiuser system without such backup. Another good lesson was bus design. We started with one of our own designs, but learned that it's important not only to find a bus that is powerful, but also one that has good support and a strong future to serve tomorrow's needs. We think the VMEbus is the only design that meets both criteria and thus have made it a standard feature of every Stride 400 Series machine."



"The marketing pressure to be compatible instead of being better, has blinded the industry."

Q: What are some of the other unique features of the 400 Series?

RC: "A surprising feature is compatibility. Everybody talks about it, but nobody does anything about it. Our systems are completely compatible with each other from the 420 model starting at $2900, through the 440, on to the powerful 460 which tops out near $60,000. Each system can talk to the others via the standard built-in local area network. Go ahead and compare this with others in the industry. You'll find their little machines don't talk to their big ones, or that the networking and multiuser are incompatible, or that they have different processors or operating systems, and so on."

Q: When you were still known as Sage Computer, you had a reputation for performance, is that still the case with the new Stride 400 Series?

RC: "Certainly, that's our calling card: 'Performance By Design.' Our new systems are actually faster; our standard processor is a 10 MHz 68000 running with no wait states. That gives us a 25% increase over the Sage models. And, we have a 12 MHz processor as an option. Let me add that speed isn't the only way to judge performance. I think it is also measured in our flexibility. We support a dozen different operating systems, not just one. And our systems service a wide variety of applications from the garage software developer to the corporate consumer running high volume business applications."

Q: Isn't that the same thing all manufacturers say in their ads?

RC: "Sure it is. But to use another over used-term, 'shop around'. We like to think of our systems as 'full service 68000 supermicrocomputers.' Take a look at everyone else's literature and then compare. When you examine cost, performance, flexibility, and utility, we don't think there's anyone else in the race. Maybe that's why we've shipped and installed more multiuser 68000 systems than anyone else."

STRIDE MICRO

Formerly Sage Computer

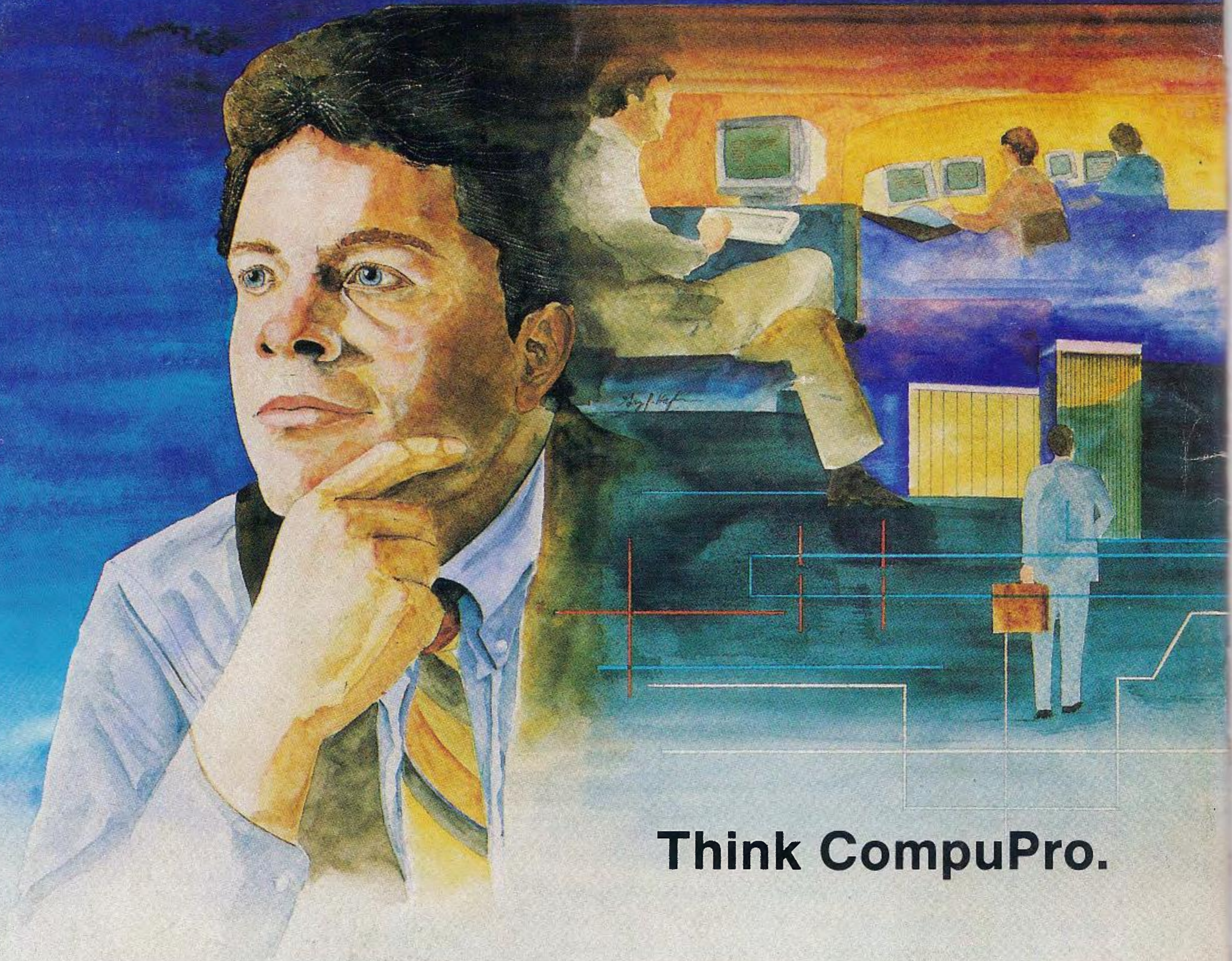
For more information on Stride or the location of the nearest Stride Dealer call or write us today. We'll also send you a free copy of our 32 page product catalog.

Corporate Offices:
4905 Energy Way
Reno, NV 89502
(702) 322-6868

Regional Offices:
Boston: (617) 229-6868
Dallas: (214) 392-7070

CIRCLE 72 ON READER SERVICE CARD

THINKING COMPUTERS?



Think CompuPro.

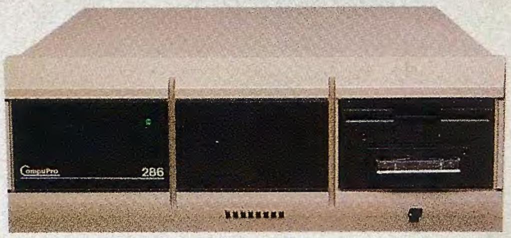
CompuPro
10PLUS

CompuPro 10*PLUS*

"... a coherent, relatively easy to use [multi-user] system with an impressive amount of power for the money and good support... alternative to stand alone—or even networked—computers, the CompuPro 10 has a lot going for it."

Personal Computing, 12/84

$7995.00 suggested list with 40 Mb hard disk



CompuPro 286

A faster version of our multi-user Model C, of which a recent review states "... hardware is high quality ... large software base... hundreds of serious business programs... versatility and adaptability to future hardware and software developments are unmatched..."

Business Computer Systems, 9/84

$9995.00 suggested list with 40Mb hard disk and cartridge tape

CIRCLE 91 ON READER SERVICE CARD

To arrange a demonstration, call **1-800-367-7816** (in California, **1-415-786-0909**).

©Viasyn 1985

Viasyn Corporation, 3506 Breakwater Court, Hayward, CA 94545